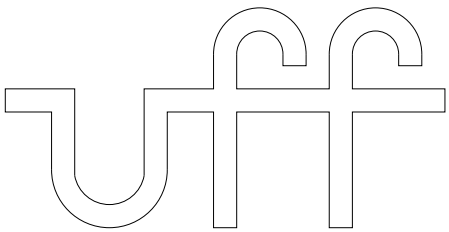
Apostila de Teoria para Técnicas Digitais II
(Versão 2k100525)



Universidade Federal Fluminense

Apostila
do
Departamento de Engenharia de Telecomunicações
da
Universidade Federal Fluminense
por
Alexandre Santos de la Vega

Maio, 2010.

.

621.3192 mudar! D278 mudar! 2010	de la Vega, Alexandre Santos Apostila de teoria para Técnicas Digitais II / Alexandre Santos de la Vega. – Niterói: UFF/CTC/TCE/TET, 2010. 123p. (atualizar...) Apostila de teoria – Graduação, Engenharia de Telecomunicações, UFF/CTC/TCE/TET, 2010. 1. Circuitos Digitais. 2. Técnicas Digitais. 3. Telecomunicações. I. Título.

Aos meus alunos.

Prefácio

O trabalho em questão cobre os tópicos abordados na disciplina Técnicas Digitais II.

A apostila foi escrita com o intuito de servir como uma referência rápida para os alunos do curso de graduação em Engenharia de Telecomunicações (TET) da Universidade Federal Fluminense (UFF).

O material básico utilizado foram as minhas notas de aula que, por sua vez, originaram-se em uma coletânea de livros sobre os assuntos abordados.

A motivação principal foi a de aumentar o dinamismo das aulas. Portanto, deve ficar bem claro que esta apostila não pretende substituir os livros textos ou outros livros de referência. Muito pelo contrário, ela deve ser utilizada apenas como ponto de partida para estudos mais aprofundados, utilizando-se a literatura existente.

Espero conseguir manter o presente texto em constante atualização e ampliação.

Correções e sugestões são sempre benvindas.

Rio de Janeiro, 19 de março de 2007.

Alexandre Santos de la Vega
TET / UFF

Agradecimentos

Aos professores do Departamento de Engenharia de Telecomunicações da Universidade Federal Fluminense (TET/UFF), que colaboraram com críticas e sugestões bastante úteis à finalização deste trabalho. Em particular, à professora Carmen Maria Costa de Carvalho, e à professora Jacqueline Silva Pereira, pela leitura meticulosa da versão original

Aos funcionários do TET/UFF, Carmen Lúcia, Jussara, Eduardo, Arlei e Francisco, pelo apoio constante.

Aos meus alunos, que, além de servirem de motivação principal, obrigam-me sempre a tentar melhorar, em todos os sentidos.

Mais uma vez, e sempre, aos meus pais, por tudo.

Rio de Janeiro, 19 de março de 2007.
Alexandre Santos de la Vega
TET / UFF

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

- Esta é uma versão inicial da apostila.
- Ela consta de tópicos desenvolvidos em sala de aula.
- Na preparação das aulas foram utilizados os seguintes livros:
  - Livros indicados pelo professor: [HP81], [Rhy73], [Uye02].
  - Livros indicados pela ementa da disciplina: [Tau82].
- Este documento aborda os seguintes assuntos:
  - Conceitos básicos: busca contextualizar a disciplina no âmbito do curso de graduação e apresentar os conceitos que serão necessários ao longo do texto.
  - Elementos básicos de armazenamento: apresenta os elementos de armazenamento utilizados nos circuitos seqüenciais abordados neste texto.
  - Circuitos seqüenciais do tipo *clock-mode*: define as características dessa classe de circuitos e aborda os procedimentos, as técnicas e as ferramentas de análise e de projeto para circuitos da classe.
  - Circuitos seqüenciais do tipo *pulsed*: define as características dessa classe de circuitos e aborda os procedimentos, as técnicas e as ferramentas de análise e de projeto para circuitos da classe.
  - Circuitos seqüenciais do tipo *level-mode*: define as características dessa classe de circuitos e aborda os procedimentos, as técnicas e as ferramentas de análise e de projeto para circuitos da classe.
  - Minimização de tabelas de estados: define o problema e apresenta técnicas de minimização para tabelas de estados.
  - Introdução à linguagem VHDL: trata de aspectos básicos da linguagem.

Capítulo 2

Circuitos seqüenciais: conceitos básicos

2.1 Introdução

- Circuitos combinacionais × circuitos seqüenciais.
- Circuitos combinacionais são sistemas instantâneos ou sem memória.
- Circuitos seqüenciais são sistemas dinâmicos ou com memória.
- Por serem sistemas instantâneos, os circuitos combinacionais respondem sempre da mesma forma, em qualquer momento, para os mesmos valores das variáveis de entrada.
- Por sua vez, por serem sistemas dinâmicos, dependendo da informação que se encontre armazenada, os circuitos seqüenciais podem responder de formas diferentes, em diferentes momentos, para os mesmos valores das variáveis de entrada.
- Circuitos seqüenciais também podem ser denominados de máquinas de estados ou de autômatos.

2.2 Estados e variáveis de estado

- Uma vez que eles são capazes de armazenar energia, os sistemas dinâmicos podem apresentar diversas configurações energéticas diferentes, denominadas **estados**.
- Uma medida do estado de um sistema, em um instante de tempo $t = t_n$, são os valores assumidos por todas as variáveis do sistema, em $t = t_n$.
- Interpretando-se o conjunto de todas as variáveis de um sistema como um espaço vetorial, pode-se selecionar um conjunto mínimo de variáveis para formar uma base para esse espaço. Uma vez que, a partir da base, podem ser obtidas todas as demais variáveis e, portanto, pode-se caracterizar o estado do sistema, as variáveis da base são denominadas **variáveis de estado** do sistema.
- Dessa forma, uma definição clássica para estado e variáveis de estado é: "O estado de um sistema, em qualquer instante de tempo $t = t_n$, é o menor conjunto de variáveis (denominadas variáveis de estado), calculadas em $t = t_n$, suficiente para determinar o comportamento do sistema para qualquer instante de tempo $t \geq t_n$, quando a entrada do sistema é conhecida para $t \geq t_n$".

2.3 Tipos de variáveis e sua interações

- Será considerado que todas as variáveis do circuito são booleanas.
- Assim sendo, os valores das variáveis podem ser interpretados como:
  - Nível: a informação é representada pelos níveis lógicos das variáveis boolenas (0 e 1). Cada nível representa um evento.
  - Borda: a informação é associada à seqüência de níveis 0 e 1 (borda positiva) ou à seqüência de níveis 1 e 0 (borda negativa). Cada borda representa um evento.
  - Transição: a informação é associada à troca de níveis 0 para 1 (transição positiva) ou à troca de níveis 1 para 0 (transição negativa). Cada transição representa um evento.
  - Pulso: a informação é associada à seqüência de níveis 0 e 1 e 0 (pulso positivo) ou à seqüência de níveis 1 e 0 e 1 (pulso negativo). A duração do valor intermediário da seqüência é denominada de largura do pulso (*pulsewidth*) e deve ser pequena em relação aos tempos envolvidos. Cada pulso representa um evento.
- Para alguns tipos de circuitos, as interações entre sinais dos tipos nível e pulso são de particular interesse. A Tabela 2.1 resume as possíveis interações, considerando-se as operações lógicas AND e OR. Os resultados indicam que, para tais operações, alguns tipos de interações produzem resultados indeterminados. Portanto, no projeto de sistemas com sinais pulsados, tais resultados devem ser levados em consideração.

A	B	$A \cdot B$	$A + B$
Nível	Nível	Nível	Nível
Nível	Pulso Positivo	Pulso Positivo	Indeterminado
Pulso Positivo	Pulso Positivo	Indeterminado	Pulso Positivo
Nível	Pulso Negativo	Indeterminado	Pulso Negativo
Pulso Negativo	Pulso Negativo	Pulso Negativo	Indeterminado
Pulso Positivo	Pulso Negativo	Indeterminado	Indeterminado

Tabela 2.1: Tipos de interações entre sinais dos tipos nível e pulso.

2.4 Modelo genérico para circuitos seqüenciais

- Na Figura 2.1 é apresentado um modelo genérico para circuitos seqüenciais, onde:
  - $x_i \in \boldsymbol{x}$, $i = 1, 2, \cdots, L$, são as variáveis de entrada ou variáveis de entrada principais.
  - $z_i \in \boldsymbol{z}$, $i = 1, 2, \cdots, M$, são as variáveis de saída ou variáveis de saída principais.
  - $Y_i \in \boldsymbol{Y}$, $i = 1, 2, \cdots, P$, são as variáveis de excitação ou variáveis de saída secundárias.
  - $y_i \in \boldsymbol{y}$, $i = 1, 2, \cdots, R$, são as variáveis de estado ou variáveis de entrada secundárias.
  - t_n é o instante atual, t_{n-1} é o instante anterior e t_{n+1} é o próximo instante.
  - $z_i^n = f_i(x_1^n, \cdots, x_L^n, y_1^n, \cdots, y_R^n)$, $i = 1, 2, \cdots, M$.
  - $Y_j^n = f_j(x_1^n, \cdots, x_L^n, y_1^n, \cdots, y_R^n)$, $j = 1, 2, \cdots, P$.
  - $y_k^{n+1} = f_k(Y_1^n, \cdots, Y_P^n)$, $k = 1, 2, \cdots, R$.
- O conjunto das variáveis y_i é denominado estado atual.
- Por sua vez, conjunto das variáveis x_i e y_i é dito estado atual total.
- O bloco denominado **Função Combinacional** é um circuito combinacional que, de acordo com a viabilidade de custo, pode ser implementado através de portas lógicas individuais, memórias ROM (*Read-Only Memory*) ou circuitos PLA (*Programmable Logic Array*).
- O bloco de memória denominado **Geração e Armazenamento das Variáveis de Estado** representa um dispositivo genérico de memória (*flip-flop*, banco de memória, atrasos de propagação).
- A função do bloco de memória não é simplesmente armazenar Y_i^n na forma de y_i^{n+1}. Pelo contrário, a sua função é mais complexa: a partir de alguns Y_i^n deve ser gerado y_j^{n+1}, o qual, então, será retido (armazenado).

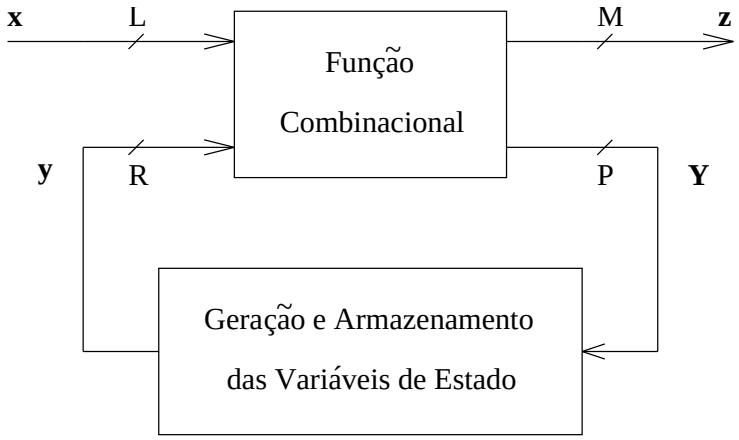


Figura 2.1: Modelo genérico para circuitos seqüenciais.

2.5 Classificação de circuitos seqüenciais quanto à dependência do sinal de saída

- Máquinas (circuitos) de Mealy e de Moore.

- Máquinas de Mealy: $z_i^n = f_i(x_1^n, \cdots, x_L^n, y_1^n, \cdots, y_R^n)$, $i = 1, 2, \cdots, M$.

- Máquinas de Moore: $z_i^n = f_i(y_1^n, \cdots, y_R^n)$, $i = 1, 2, \cdots, M$.

- As Figuras 2.2 e 2.3 apresentam, respectivamente, um exemplo de máquina de Mealy e um exemplo de máquina de Moore.

- Geralmente, as máquinas de Mealy são implementadas por circuitos mais simples do que as máquinas de Moore.

- Por outro lado, nas máquinas de Moore, em conseqüência de sua definição, os valores dos sinais de saída permanecem constantes entre dois estados consecutivos. Portanto, torna-se mais simples controlar a interação entre diversos blocos de circuitos desse tipo. Pela mesma razão, é mais fácil acompanhar a evolução dos estados do circuito, o que simplifica a depuração de erros.

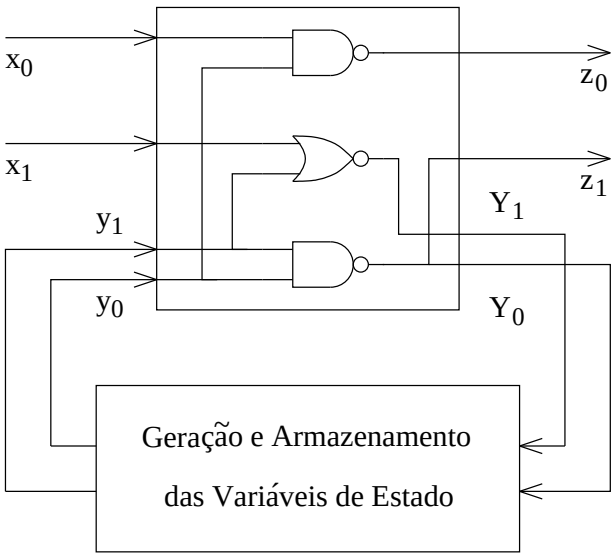


Figura 2.2: Exemplo de máquina de Mealy.

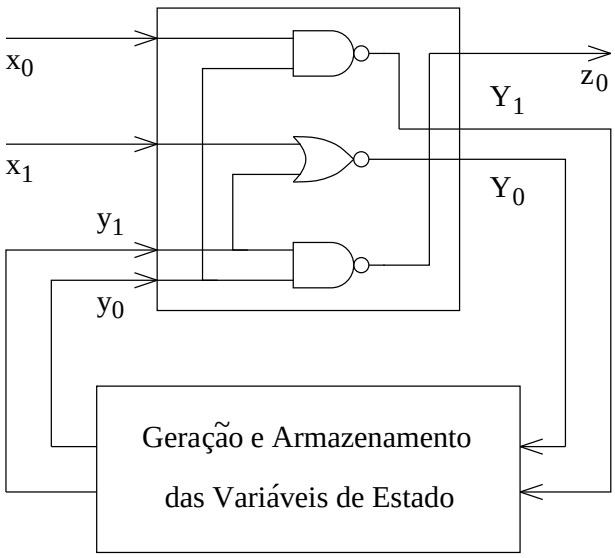


Figura 2.3: Exemplo de máquina de Moore.

2.6 Classificação de circuitos seqüenciais quanto ao tipo de controle da mudança de estado

- Na literatura, são encontradas várias denominações diferentes para designar os diversos tipos de circuitos seqüenciais existentes.
- A nomenclatura aqui utilizada será a seguinte:
  - *Clock-mode* ou *clocked*.
  - *Pulsed*.
  - *Level-mode*.

2.6.1 Circuitos seqüenciais *clock-mode* ou *clocked*

- A Figura 2.4 ilustra um modelo genérico para circuitos seqüenciais *clock-mode*.
- Todas as variáveis carregam informação nos níveis.
- As variáveis de estado são modificadas apenas pela ação de um sinal pulsante, com função de temporização ou de controle, comumente denominado de relógio (*clock*).
- Apesar de ser um sinal pulsante, não é necessário que o *clock* seja periódico.
- O sinal de *clock* não carrega qualquer tipo de informação. Ele só determina quando haverá mudança de estado.
- As variáveis de excitação, em conjunto com os elementos de armazenamento, determinam qual será a mudança de estado.
- As variáveis de entrada devem estar estáveis quando da atuação do *clock*.

- Um *clock* atuando em t_n, com $\boldsymbol{x}^n$, $\boldsymbol{z}^n$, e $\boldsymbol{Y}^n$ estáveis, provoca uma mudança de estado de $\boldsymbol{y}^n$ para $\boldsymbol{y}^{n+1}$.
- O circuito deve estar estável entre dois pulsos de *clock*. Assim, o que limita a freqüência máxima de operação do circuito é, basicamente, a soma do tempo de estabilização da memória com o tempo de propagação máximo do circuito combinacional.
- De certa forma, um circuito seqüencial *clock-mode* pode ser interpretado como um caso particular de circuitos seqüenciais *pulsed*.

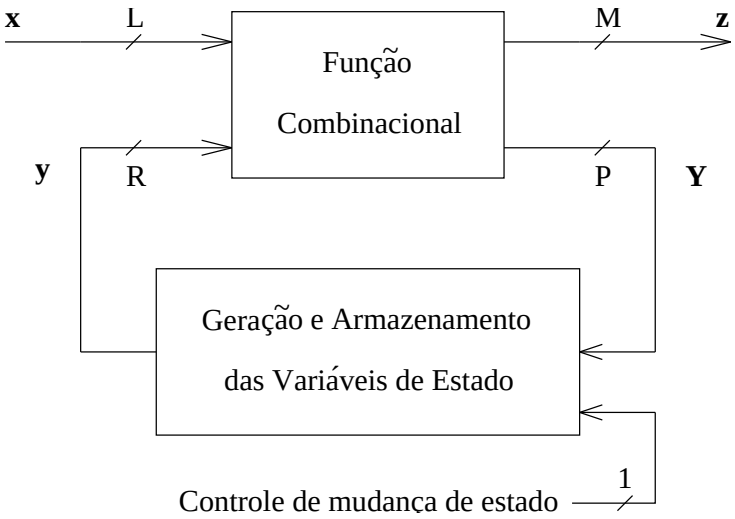


Figura 2.4: Modelo genérico para circuitos seqüenciais *clock-mode*.

2.6.2 Circuitos seqüenciais *pulsed*

- A Figura 2.5 apresenta um modelo genérico para circuitos seqüenciais *pulsed*.
- Não há um sinal pulsante de *clock* separado, sem informação.
- A mudança de estado ocorre pela atuação de um pulso em um sinal de entrada.

2.6.3 Circuitos seqüenciais *level-mode*

- Na Figura 2.6 pode ser visto um modelo genérico para circuitos seqüenciais *level-mode*.
- A realimentação das variáveis de excitação Y_i, gerando as variáveis de estado y_j, é realizada de forma contínua, ao contrário das demais classes, onde a mesma é controlada.
- A mudança de estado ocorre pela atuação de níveis dos sinais de entrada.
- Caso particular: operação em modo fundamental, onde uma mudança de nível só pode ocorrer após a mudança de nível anterior ter levado a máquina a um estado estável.
- Assim como nos demais classes: $y_k^{n+1} = f_k(Y_1^n, \cdots, Y_P^n)$, $k = 1, 2, \cdots, R$.
- Mais especificamente, neste caso: $P = R$ e $y_k(t + \Delta t_k) = Y_k(t)$, $k = 1, 2, \cdots, P$.

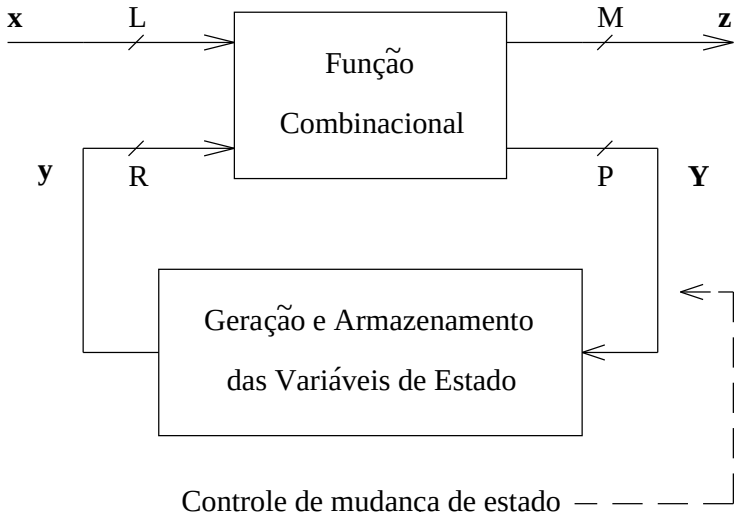


Figura 2.5: Modelo genérico para circuitos seqüenciais *pulsed*.

- Os atrasos Δt_k que implementam o bloco de memória não são blocos de retardo isolados. Eles representam a concentração de atrasos de propagação existentes no circuito combinacional.

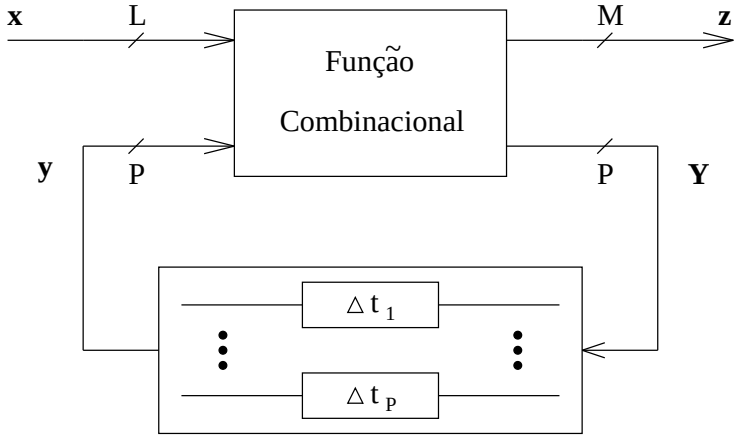


Figura 2.6: Modelo genérico para circuitos seqüenciais *level-mode*.

Capítulo 3

Elementos básicos de armazenamento

3.1 Introdução

- Se toda a informação presente em um circuito seqüencial for expressa por meio de valores binários, os elementos básicos de armazenamento deverão ser dispositivos capazes de armazenar variáveis booleanas.

- Assim, os requisitos básicos para tais dispositivos são:

  - Capacidade de representar os valores lógicos "0" e "1".
  - Possibilidade de representar apenas os valores lógicos "0" e "1".
  - Capacidade de travar (*latch*) os valores lógicos "0" e "1" por tempo indeterminado.
  - Capacidade de decidir sobre o valor lógico a ser armazenado, a partir de sinais de acionamento.

- Os requisitos acima definem um dispositivo com dois estados, estáveis, cuja mudança de estados é disparada (*triggered*) por sinais de ativação específicos.

- Tecnicamente, tal dispositivo é denominado de multivibrador biestável.

- Popularmente, embora não haja um consenso sobre a classificação dos dispositivos, são empregadas as denominações *latch* e *flip-flop*.

- Um dispositivo multivibrador biestável pode ser implementado através de circuitos analógicos, utilizando-se transistores, resistores e capacitores.

- Por outro lado, é possível obter uma implementação dita digital, utilizando-se apenas portas lógicas como elementos primitivos.

- Do ponto de vista de integração do sistema (lógica combinacional + lógica seqüencial), a implementação digital pode ser interpretada como a mais adequada para o projeto de sistemas digitais, uma vez que utiliza portas lógicas como elementos primitivos.

- Deve ser ressaltado que, pela sua própria caracterização, os elementos básicos de armazenamento, implementados de forma digital, são circuitos seqüenciais elementares, do tipo *level-mode*.

3.2 Classificação quanto à funcionalidade

- No tocante à funcionalidade, existem quatro tipos básicos de *flip-flops*: SR, JK, D e T.

- Dependendo do tipo de implementação do dispositivo e dos sinais de ativação existentes, diversas variações desses quatro tipos básicos podem ser definidas e implementadas.

- Independentemente das possíveis variações, a funcionalidade básica de cada um dos quatro tipos citados pode ser representada pelas seguintes equações, onde X^n representa o valor da variável X no instante t_n e X^{n+1} representa o valor da variável X no instante seguinte t_{n+1}:

  - *Flip-flop* SR:

$$\begin{cases} Q^{n+1} = (S^n) + \left(\overline{R^n} \cdot Q^n\right) & , \text{ para } (S^n \cdot R^n) = 0 \\ Indeterminado & , \text{ para } S^n = R^n = 1 \end{cases} . \tag{3.1}$$

  - *Flip-flop* JK:

$$Q^{n+1} = \left(J^n \cdot \overline{Q^n}\right) + \left(\overline{K^n} \cdot Q^n\right) . \tag{3.2}$$

  - *Flip-flop* D:

$$Q^{n+1} = D^n . \tag{3.3}$$

  - *Flip-flop* T_1:

$$Q^{n+1} = \overline{Q^n} \tag{3.4}$$

  - *Flip-flop* T_2:

$$Q^{n+1} = \left(T^n \cdot \overline{Q^n}\right) + \left(\overline{T^n} \cdot Q^n\right) . \tag{3.5}$$

- As operações básicas, associadas às Equações (3.1), (3.2), (3.3) e (3.5), podem ser mais facilmente identificadas através de suas respectivas tabelas, apresentadas na Figura 3.1.

- As variáveis S, R, J, K, D e T representam os sinais de entrada, enquanto a variável Q representa o sinal de saída dos respectivos *flip-flops*.

- Das equações apresentadas, e de suas respectivas tabelas, torna-se natural o significado da nomenclatura dos sinais: Q (*Quiescent*), SR (*Set-Reset*), D (*unit Delay*) e T (*Toggle*).

- A nomenclatura JK surgiu historicamente, sem qualquer relação com a sua funcionalidade.

S^n	R^n	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	proibido

J^n	K^n	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	$\overline{Q^n}$

D^n	Q^{n+1}
0	0
1	1

T^n	Q^{n+1}
0	Q^n
1	$\overline{Q^n}$

Figura 3.1: Tabelas de operação básica para os *flip-flops* SR, JK, D e T_2.

3.3 Relacionamento entre os tipos básicos de *flip-flops*

- Observando-se as equações dos tipos básicos de *flip-flops*, e suas respectivas tabelas, pode-se notar um estreito relacionamento entre eles.
- Alguns desses relacionamentos podem ser estabelecidos sem o emprego de realimentação, o que acontece nos casos de um *flip-flop* com mais funcionalidade para um *flip-flop* com menos funcionalidade.
- Os casos contrários requerem que o *flip-flop* seja realimentado.
- Inicialmente, pode-se estabelecer as seguintes relações entre os *flip-flops* SR, JK, D e T:
  - Para as combinações de entrada "00", "01" e "10", os *flip-flops* SR e JK possuem o mesmo comportamento.
  - O *flip-flop* JK amplia a operação do *flip-flop* SR, implementando uma funcionalidade para a combinação de entrada "11".
  - O *flip-flop* JK, com as entradas $J = K = 1$ ou $J = K = T$, é equivalente ao *flip-flop* T, de acordo com as Equações (3.4) e (3.5), respectivamente.
  - Por sua vez, um *flip-flop* D pode ser implementado a partir de *flip-flops* SR ou JK, se $S = D$ e $R = \overline{S}$ ou se $J = D$ e $K = \overline{J}$, respectivamente.
  - Um *flip-flop* T_1 pode ser implementado a partir de um *flip-flop* T_2, fazendo-se $T = 1$.
  - A partir de um *flip-flop* D pode-se implementar um *flip-flop* T, adotando-se $D = \overline{Q}$ ou $D = \left(T \cdot \overline{Q}\right) + \left(\overline{T} \cdot Q\right)$, conforme as Equações (3.4) e (3.5), respectivamente.
- A Tabela 3.1 apresenta um resumo de transformações envolvendo *flip-flops* dos tipos SR, JK, D, e T, utilizando suas entradas e saídas como variáveis de projeto. As opções marcadas com (*) indicam a impossibilidade desse tipo de projeto, uma vez que o *flip-flop* do tipo T_1 não possui entrada de dados. Existe apenas um sinal de sincronismo ($CTRL$ ou CK) que controla a sua operação. Sendo assim, uma solução diferente deve ser proposta, a qual atue sobre tal sinal de controle.

Transformação desejada	Tipo de arquitetura	
	Sem realimentação	Com realimentação
$JK \longrightarrow SR$	Não aplicar: $J = K = 1$	—
$JK \longrightarrow D$	$J = D$; $K = \overline{J}$	—
$JK \longrightarrow T_1$	$J = K = 1$	—
$JK \longrightarrow T_2$	$J = K = T$	—
$SR \longrightarrow JK$	—	$S = (J \cdot \overline{Q})$; $R = (K \cdot Q)$
$D \longrightarrow JK$	—	$D = (J \cdot \overline{Q}) + (\overline{K} \cdot Q)$
$T_1 \longrightarrow JK$	(*)	(*)
$T_2 \longrightarrow JK$	—	$T = (J \cdot \overline{Q}) + (K \cdot Q)$
$SR \longrightarrow D$	$S = D$; $R = \overline{S}$	—
$SR \longrightarrow T_1$	—	$S = \overline{Q}$; $R = Q$
$SR \longrightarrow T_2$	—	$S = (T \cdot \overline{Q})$; $R = (T \cdot Q)$
$D \longrightarrow SR$	—	$D = (S) + (\overline{R} \cdot Q)$
$T_1 \longrightarrow SR$	(*)	(*)
$T_2 \longrightarrow SR$	—	$T = (S \cdot \overline{Q}) + (R \cdot Q)$
$D \longrightarrow T_1$	—	$D = \overline{Q}$
$D \longrightarrow T_2$	—	$D = (T \cdot \overline{Q}) + (\overline{T} \cdot Q)$
$T_1 \longrightarrow D$	(*)	(*)
$T_2 \longrightarrow D$	—	$T = (D \cdot \overline{Q}) + (\overline{D} \cdot Q)$
$T_1 \longrightarrow T_2$	(*)	(*)
$T_2 \longrightarrow T_1$	$T = 1$	—

Tabela 3.1: Transformações envolvendo *flip-flops* dos tipos JK, D, T_1 e T_2.

3.4 Mapas de excitação dos *flip-flops*

- Uma outra forma de descrever a operação de um *flip-flop* é através do tipo de excitação que deve ser aplicado nas suas entradas a fim de provocar uma determinada variação na sua saída. Tal forma de descrição é denominada mapa de excitação.

- A Figura 3.2 apresenta os mapas de excitação para os *flip-flops* SR, JK, D e T_2.

Q^n	$\rightarrow$	Q^{n+1}	S^n	R^n
0	$\rightarrow$	0	0	X
0	$\rightarrow$	1	1	0
1	$\rightarrow$	0	0	1
1	$\rightarrow$	1	X	0
0	$\rightarrow$	X	X	X
1	$\rightarrow$	X	X	X

Q^n	$\rightarrow$	Q^{n+1}	J^n	K^n
0	$\rightarrow$	0	0	X
0	$\rightarrow$	1	1	X
1	$\rightarrow$	0	X	1
1	$\rightarrow$	1	X	0
0	$\rightarrow$	X	X	X
1	$\rightarrow$	X	X	X

Q^n	$\rightarrow$	Q^{n+1}	D^n
0	$\rightarrow$	0	0
0	$\rightarrow$	1	1
1	$\rightarrow$	0	0
1	$\rightarrow$	1	1
0	$\rightarrow$	X	X
1	$\rightarrow$	X	X

Q^n	$\rightarrow$	Q^{n+1}	T^n
0	$\rightarrow$	0	0
0	$\rightarrow$	1	1
1	$\rightarrow$	0	1
1	$\rightarrow$	1	0
0	$\rightarrow$	X	X
1	$\rightarrow$	X	X

Figura 3.2: Mapas de excitação para os *flip-flops* SR, JK, D e T_2.

3.5 Tipos de comportamento das saídas dos *flip-flops*

- Os tipos de comportamento que a saída de um *flip-flop* pode apresentar, de um instante de tempo (t_n) para o instante de tempo seguinte (t_{n+1}), são definidos na Tabela 3.2.

Q^n	$\rightarrow$	Q^{n+1}	Símbolo	Tipo de Comportamento
0	$\rightarrow$	0	0	Estático
0	$\rightarrow$	1	α	Dinâmico
1	$\rightarrow$	0	β	Dinâmico
1	$\rightarrow$	1	1	Estático
0	$\rightarrow$	X	X	Indeterminado
1	$\rightarrow$	X	X	Indeterminado

Tabela 3.2: Definição dos tipos de comportamento apresentados pela saída de um *flip-flop*.

3.6 Excitação × comportamento

- As tabelas da Figura 3.3 associam os tipos de comportamento da saída às respectivas excitações que as entradas devem sofrer, para os *flip-flops* SR, JK, D e T_2.

$Q^n \rightarrow Q^{n+1}$	Variação	S^n	R^n
$0 \rightarrow 0$	0	0	X
$0 \rightarrow 1$	α	1	0
$1 \rightarrow 0$	β	0	1
$1 \rightarrow 1$	1	X	0
$0 \rightarrow X$	X	X	X
$1 \rightarrow X$	X	X	X

$Q^n \rightarrow Q^{n+1}$	Variação	J^n	K^n
$0 \rightarrow 0$	0	0	X
$0 \rightarrow 1$	α	1	X
$1 \rightarrow 0$	β	X	1
$1 \rightarrow 1$	1	X	0
$0 \rightarrow X$	X	X	X
$1 \rightarrow X$	X	X	X

$Q^n \rightarrow Q^{n+1}$	Variação	D^n
$0 \rightarrow 0$	0	0
$0 \rightarrow 1$	α	1
$1 \rightarrow 0$	β	0
$1 \rightarrow 1$	1	1
$0 \rightarrow X$	X	X
$1 \rightarrow X$	X	X

$Q^n \rightarrow Q^{n+1}$	Variação	T^n
$0 \rightarrow 0$	0	0
$0 \rightarrow 1$	α	1
$1 \rightarrow 0$	β	1
$1 \rightarrow 1$	1	0
$0 \rightarrow X$	X	X
$1 \rightarrow X$	X	X

Figura 3.3: Tipos de comportamento e respectivas excitações para os *flip-flops* SR, JK, D e T_2.

3.7 Funcionalidade × excitação × comportamento

- A Tabela 3.3 apresenta um resumo geral de funcionalidade-excitação-comportamento, relacionando os valores de excitação a serem aplicados nas entradas, a partir de cada tipo de comportamento da saída, para cada tipo de *flip-flop*.

Entrada	Entrada = "1"	Entrada = "0"	Entrada = "X"
S	α	0 , β	1 , X
R	β	1 , α	0 , X
J	α	0	1 , β , X
K	β	1	0 , α , X
D	1 , α	0 , β	X
T	α , β	0 , 1	X

Tabela 3.3: Tabela resumo de funcionalidade-excitação-comportamento para os *flip-flops* SR, JK, D e T_2.

3.8 Circuitos seqüenciais × tabelas dos *flip-flops*

- Uma vez que os *flip-flops* podem usados como elementos básicos de armazenamento nos circuitos seqüenciais, as tabelas que os definem apresentam-se como ferramentas de análise e síntese para tais circuitos.
- As aplicações e os termos citados a seguir serão definidos nos próximos capítulos.
- No processo de análise de um circuito seqüencial, as tabelas de operação dos *flip-flops* são utilizadas para montar a tabela de mudança de estados.
- No processo de síntese, as tabelas de excitação e de comportamento são necessárias para montar os mapas-K de excitação e de transição, respectivamente.
- Os mapas-K de excitação apresentam os valores que as variáveis de excitação do circuito seqüencial, que são as variáveis de entrada dos elementos de memória, devem assumir, em função das suas variáveis de estado e das variáveis de entrada. É utilizado um mapa-K específico para cada entrada de cada *flip-flop*.
- Os mapas-K de transição descrevem o comportamento dos elementos de memória do circuito seqüencial, em função das suas variáveis de estado e das variáveis de entrada. É necessário apenas um único mapa-K para todos os tipos de *flip-flops*, para cada elemento de memória.
- Portanto, as funções lógicas que geram as variáveis de excitação, que são as variáveis de entrada dos elementos de memória, podem ser obtidas: i) do mapa-K de excitação de cada entrada, de cada *flip-flop* ou ii) do mapa-K de transição de cada elemento de memória, em conjunto com a tabela resumo 3.3.
- Como exemplo, a Tabela 3.4 descreve as mudanças de estado e os tipos de comportamento dos elementos de memória para um contador binário, crescente, de três *bits*. Por sua vez, os mapas-K de transição dos elementos de memória e os mapas-K de escitação para *flip-flops* JK são apresentados na Figuras 3.4 e 3.5, respectivamente. Deve-se notar que este contador não possui variáveis de entrada. Das tabelas das Figuras 3.4 e 3.5, pode-se obter

$$J_2 = K_2 = (Q_1 \cdot Q_0) \ , \tag{3.6}$$

$$J_1 = K_1 = Q_0 \tag{3.7}$$

e

$$J_0 = K_0 = 1 \ . \tag{3.8}$$

Q_2^n	Q_1^n	Q_0^n	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}	Q_2	Q_1	Q_0
0	0	0	0	0	1	0	0	α
0	0	1	0	1	0	0	α	β
0	1	0	0	1	1	0	1	α
0	1	1	1	0	0	α	β	β
1	0	0	1	0	1	1	0	α
1	0	1	1	1	0	1	α	β
1	1	0	1	1	1	1	1	α
1	1	1	0	0	0	β	β	β

Tabela 3.4: Tabela de mudanças de estado e de comportamento dos elementos de memória para um contador binário, crescente, de três *bits*.

$FF2$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	0	α	0
1	1	1	β	1

$FF1$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	α	β	1
1	0	α	β	1

$FF0$

Q_2 \ Q_1Q_0	00	01	11	10
0	α	β	β	α
1	α	β	β	α

Figura 3.4: Mapas-K de transição para os elementos de memória de um contador binário, crescente, de três *bits*.

J_2

Q_2 \ Q_1Q_0	00	01	11	10
0	0	0	1	0
1	X	X	X	X

K_2

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	X	X
1	0	0	1	0

J_1

Q_2 \ Q_1Q_0	00	01	11	10
0	0	1	X	X
1	0	1	X	X

K_1

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	1	0
1	X	X	1	0

J_0

Q_2 \ Q_1Q_0	00	01	11	10
0	1	X	X	1
1	1	X	X	1

K_0

Q_2 \ Q_1Q_0	00	01	11	10
0	X	1	1	X
1	X	1	1	X

Figura 3.5: Mapas-K de excitação para os *flip-flops* JK de um contador binário, crescente, de três *bits*.

3.9 Estruturas estáticas simétricas

- Os elementos básicos de armazenamento (*flip-flops*) podem ser implementados de diversas formas diferentes.

- Duas características são de grande interesse para o projeto de circuitos seqüenciais: i) que os *flip-flops* possuam saídas complementares e ii) que a temporização das mudanças dos valores de tais saídas possua o maior sincronismo possível.

- Tais características podem ser obtidas através de estruturas simétricas.

- A Figura 3.6 apresenta uma estrutura simétrica de armazenamento, implementada por dois inversores autorealimentados.

- A autorealimentação confere uma característica de armazenamento estático à estrutura, pois suas saídas Q e $\overline{Q}$ estarão estáveis (quiescentes) enquanto os inversores estiverem energizados.

- A estrutura da Figura 3.6 apresenta uma grande desvantagem: não é controlável.

- Algumas propostas para tornar o circuito da Figura 3.6 controlável são ilustradas na Figura 3.7.

  – No primeiro caso, Figura 3.7.a, utiliza-se um inversor com capacidade de corrente alta (inversor forte), um inversor com capacidade de corrente baixa (inversor fraco) e uma única chave responsável pela escrita do dado binário.

  – No segundo caso, Figura 3.7.b, são utilizados dois inversores idênticos, enquanto uma chave de duas posições controla a escrita e a manutenção do dado binário.

  – No terceiro caso, Figura 3.7.c, são utilizados dois inversores idênticos e a chave de duas posições é implementada através de duas chaves com controles independentes para escrita e armazenamento.

  – No último caso, Figura 3.7.d, são utilizados dois inversores idênticos e a chave de duas posições é implementada através de duas chaves com acionamentos complementares para escrita e armazenamento.

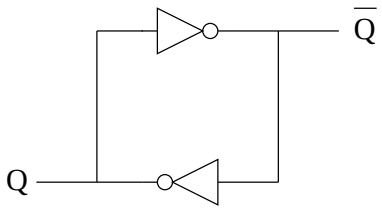


Figura 3.6: Estrutura de armazenamento estática e simétrica, não controlável.

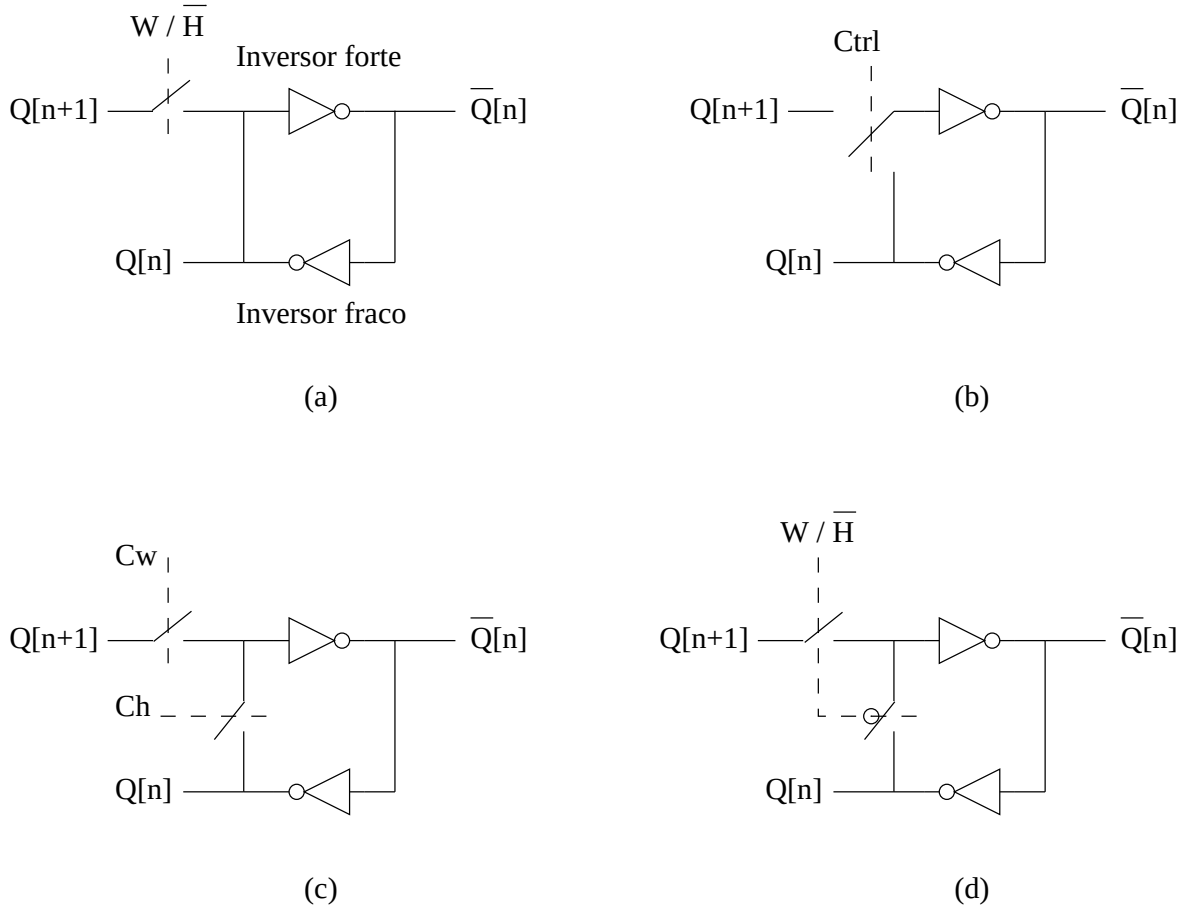


Figura 3.7: Estruturas de armazenamento estáticas e simétricas, controláveis por chaves.

3.10 Exemplos de *flip-flops*

- Uma vez que *flip-flops* são circuitos seqüenciais do tipo *level-mode*, os mesmos devem ser projetados adequadamente, por meio das técnicas existentes para tais tipos de sistemas.
- Porém, ainda que não se conheça a forma como foram projetados, não é difícil analisar o funcionamento de um determinado *flip-flop*.
- A seguir são apresentadas algumas implementações de *flip-flops*.
- Embora não haja um consenso na classificação dos *flip-flops*, os mesmos serão divididos em: *unclocked* (sem sinal de controle de sincronismo) e *clocked* (com sinal de controle de sincronismo).

3.10.1 *Flip-flops* do tipo *unclocked*

- Os *flip-flops* do tipo *unclocked* são também denominados de *latches*.
- O circuito de armazenamento estático da Figura 3.6 pode ser controlado usando apenas portas lógicas. O primeiro passo nesse sentido é substituir os inversores por portas lógicas NOR ou NAND. Em seguida, um terminal de entrada de cada porta deve ser desconectado, a fim de ser utilizado como terminal de controle (S e R). O processo é ilustrado nas Figuras 3.8 e 3.9.

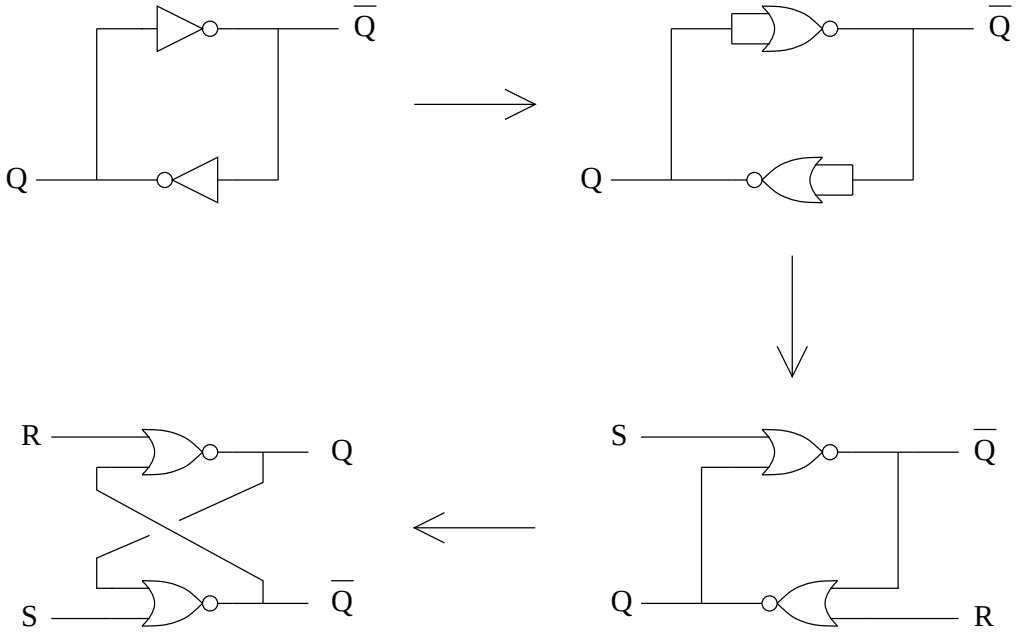


Figura 3.8: Uso de portas lógicas NOR na implementação de controle em uma estrutura de armazenamento estática e simétrica.

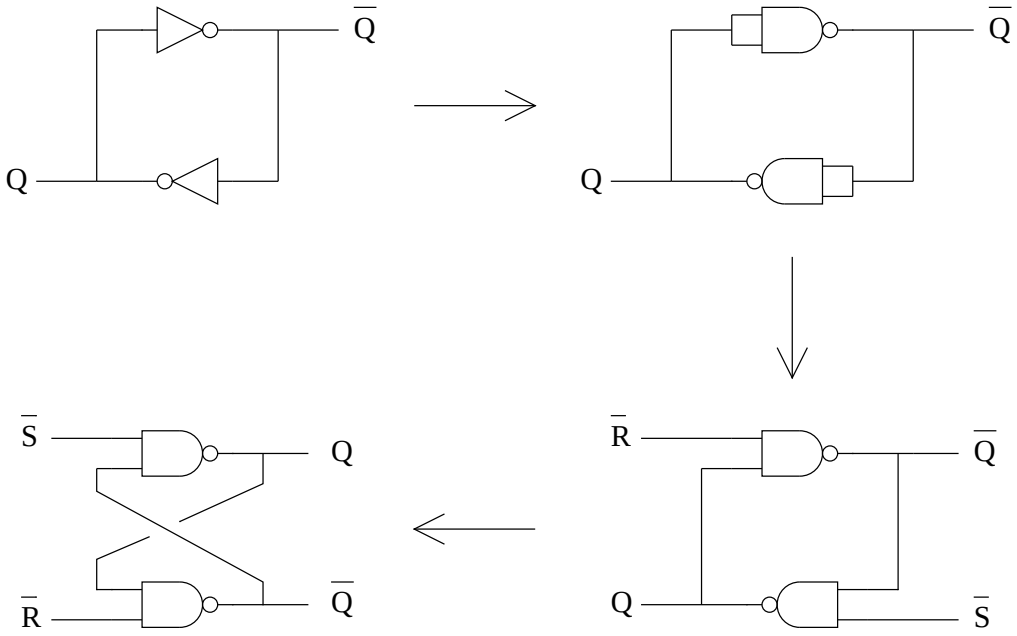


Figura 3.9: Uso de portas lógicas NAND na implementação de controle em uma estrutura de armazenamento estática e simétrica.

- Deve ser notado que, enquanto $S = R = 0$, os valores de Q e $\overline{Q}$ são mantidos estáveis.

- Alterando-se os valores dos sinais de controle para $S = 1$ e $R = 0$, obtém-se:

  - $\overline{Q}_{NOR} = 0$ e, em seguida, $Q_{NOR} = 1$.
  - $Q_{NAND} = 1$ e, em seguida, $\overline{Q}_{NAND} = 0$.

- Retornando-se à condição $S = R = 0$, os valores de Q e $\overline{Q}$ são mantidos estáveis.

- Alterando-se os valores dos sinais de controle para $S = 0$ e $R = 1$, obtém-se:

  - $Q_{NOR} = 0$ e, em seguida, $\overline{Q}_{NOR} = 1$.
  - $\overline{Q}_{NAND} = 1$ e, em seguida, $Q_{NAND} = 0$.

- Se forem atribuídos os valores $S = R = 1$, o resultado é indeterminado e não complementar. No caso da implementação com NOR, $Q_{NOR} = \overline{Q}_{NOR} = 0$. No caso da implementação com NAND, $Q_{NAND} = \overline{Q}_{NAND} = 1$. Por essa razão, tal configuração é dita proibida.

- A Tabela 3.5 resume a análise acima, de onde pode-se observar que ambos os circuitos implementam um *flip-flop* do tipo *unclocked* SR.

- Quanto aos demais tipos de *flip-flop*:

  - Acrescentando-se uma porta lógica inversora aos circuitos, de forma que $R = \overline{S}$, eles podem implementar um *flip-flop* do tipo *unclocked* D. Porém, tal construção não tem utilidade prática, uma vez que o circuito final passa a se comportar como um mero propagador do sinal de entrada, sem controle de retenção.
  - Devido a problemas de instabilidade, não é possível implementar *flip-flops* dos tipos *unclocked* JK e *unclocked* T.

- Finalmente, cabe observar que, embora o *flip-flop* do tipo *unclocked* SR possua várias limitações, o mesmo é usado como núcleo básico para a implementação dos *flip-flops* do tipo *clocked*, conforme será ilustrado a seguir.

S^n	R^n	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	proibido

Tabela 3.5: Operação das estruturas de armazenamento estáticas e simétricas controladas por meio de portas lógicas NOR e NAND.

3.10.2 *Flip-flops* do tipo *clocked*

- Dependendo da arquitetura utilizada, podem ser destacadas três classes de *flip-flops* do tipo *clocked*: elementar, *master-slave* e *edge-triggered*.

Flip-flops do tipo *clocked* elementar

- Em relação aos *flip-flops* do tipo *clocked* elementar, pode-se dizer que um SR é um *latch* com controle de sincronismo, conforme exemplificado nas Figuras 3.10 e 3.11. Por sua vez, um *flip-flop* D pode ser implementado a partir de um SR, conforme ilustrado na Figura 3.12.

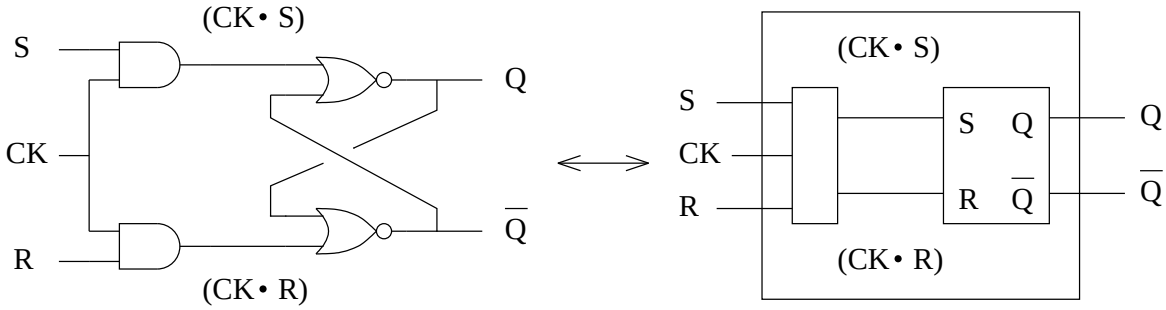


Figura 3.10: Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NOR.

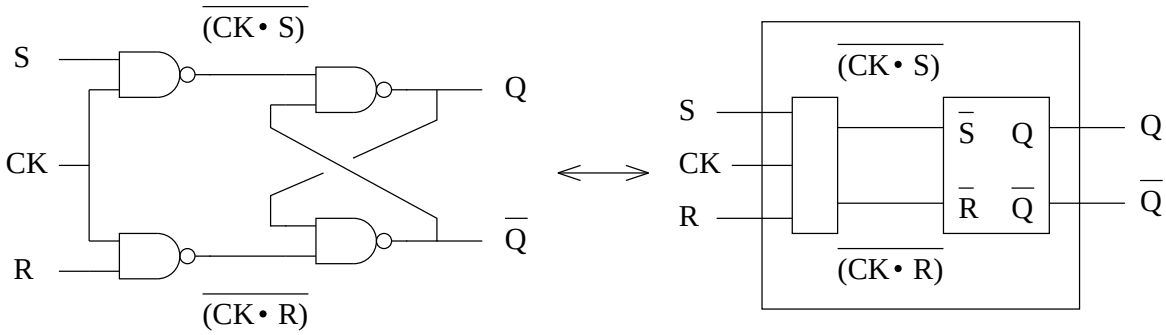


Figura 3.11: Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NAND.

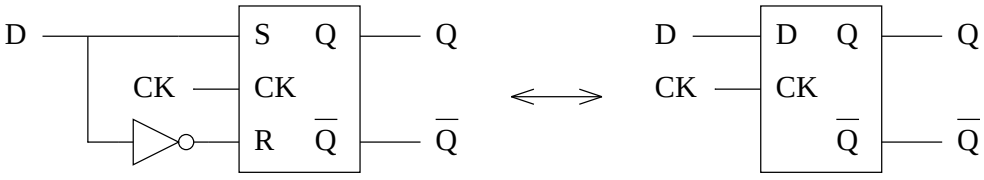


Figura 3.12: Exemplo de implementação de *flip-flop* D do tipo *clocked* elementar, com base em um *flip-flop* SR.

Flip-flops* dos tipos *clocked master-slave* e *clocked edge-triggered

- O tipo elementar pode ser usado como bloco básico de construção para outras estruturas funcionais. Os tipos *master-slave* e *edge-triggered* são soluções propostas para problemas que podem surgir em tais implementações.

- O tipo *master-slave* emprega o conceito de *pipelining*. A idéia por trás dessa técnica é que, a cada unidade funcional de uma cadeia de processamento, sejam adicionados elementos de memória de entrada (*master*) e de saída (*slave*), com sinais de controle de carregamento alternados. Dessa forma, todas as unidades da cadeia trabalham em paralelo, aumentando o fluxo de processamento (*throughput*). A técnica é ilustrada na Figura 3.13. No caso do *flip-flop master-slave*, a unidade funcional é apenas uma transmissão, conectando os elementos de memória de entrada e de saída.

- Embora uma estrutura *master-slave* empregue o dobro do circuito necessário ao armazenamento, ela permite um maior controle de fluxo entre a entrada e a saída do *flip-flop*. Uma vez que os sinais de entrada só provocam modificações na saída após uma alternância de sinais de controle, tais *flip-flops* podem ser interpretados como sensíveis a bordas (de subida ou de descida) ou a pulsos (positivo ou negativo).

- O tipo *edge-triggered* é uma solução proposta para um problema de operação apresentado pelo tipo *master-slave*. Nessa estrutura, além da célula básica de armazenamento, circuitos realimentados garantem que, logo após ocorra uma transição do sinal de controle, o *flip-flop* fique insensível a qualquer variação dos sinais de entrada, até que ocorra uma outra transição do mesmo tipo. Assim, desprezando-se o tempo necessário à insensibilização da estrutura, pode-se dizer que a mesma é sensível a transições (positiva ou negativa).

- Um exemplo de implementação para um *flip-flop* D do tipo *clocked*, com estrutura *master-slave*, pode ser encontrado na Figura 3.14, onde é empregado um *flip-flop* SR como célula básica.

- Não é difícil mostrar que um *flip-flop* SR pode ser usado para implementar um *flip-flop* JK, desde que $S = (\overline{Q} \cdot J)$ e $R = (Q \cdot K)$. Implementações utilizando *flip-flops* SR *unclocked* e *clocked* são mostradas nas Figuras 3.15 e 3.16, respectivamente. Uma vez que a realimentação das saídas (Q e $\overline{Q}$) para as entradas (J e K) é realizada de forma contínua, ambas apresentam o mesmo problema: oscilam quando $J = K = 1$. Para solucionar esse problema, exemplos de implementação para um *flip-flop* JK do tipo *clocked*, com estrutura *master-slave*, são apresentados nas Figuras 3.17 – 3.19.

- Devido a problemas de temporização, o *flip-flop* D da Figura 3.14 pode apresentar mau funcionamento e até mesmo oscilações. Uma implementação mais robusta é alcançada utilizando-se o *flip-flop* JK *master-slave*, com $D = J$ e $K = \overline{J}$.

- Por sua vez, um *flip-flop* T pode ser implementado com $J = K = 1$ ou $J = K = T$.

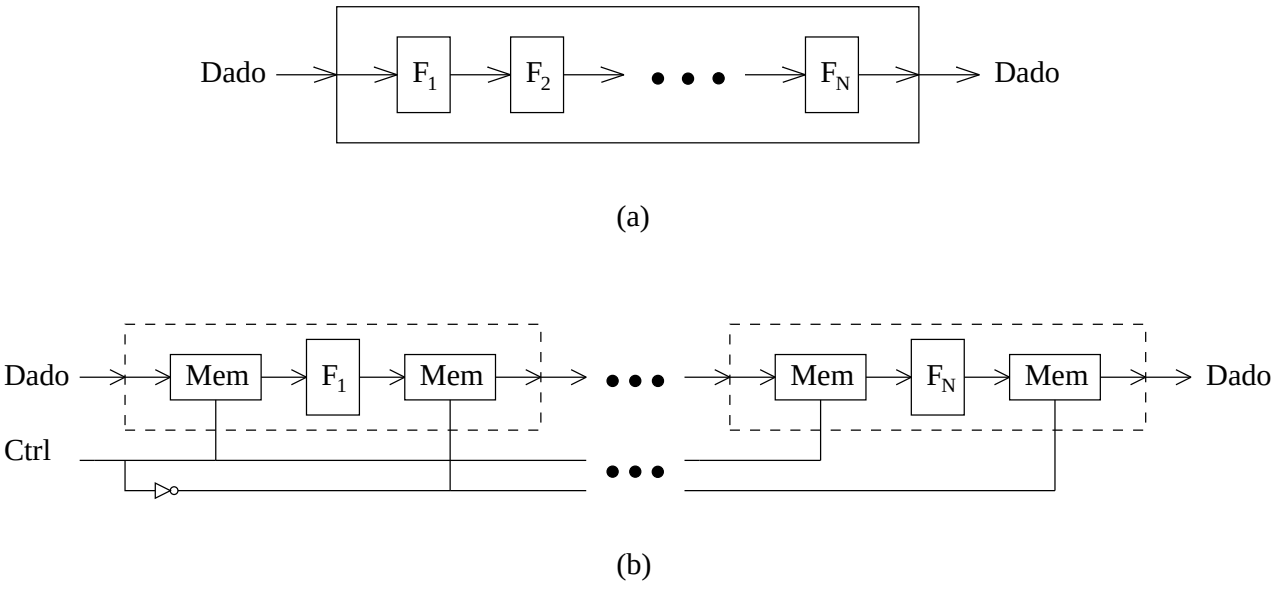


Figura 3.13: Técnica de *pipelining*: (a) Bloco funcional original e (b) Bloco com *pipelining*.

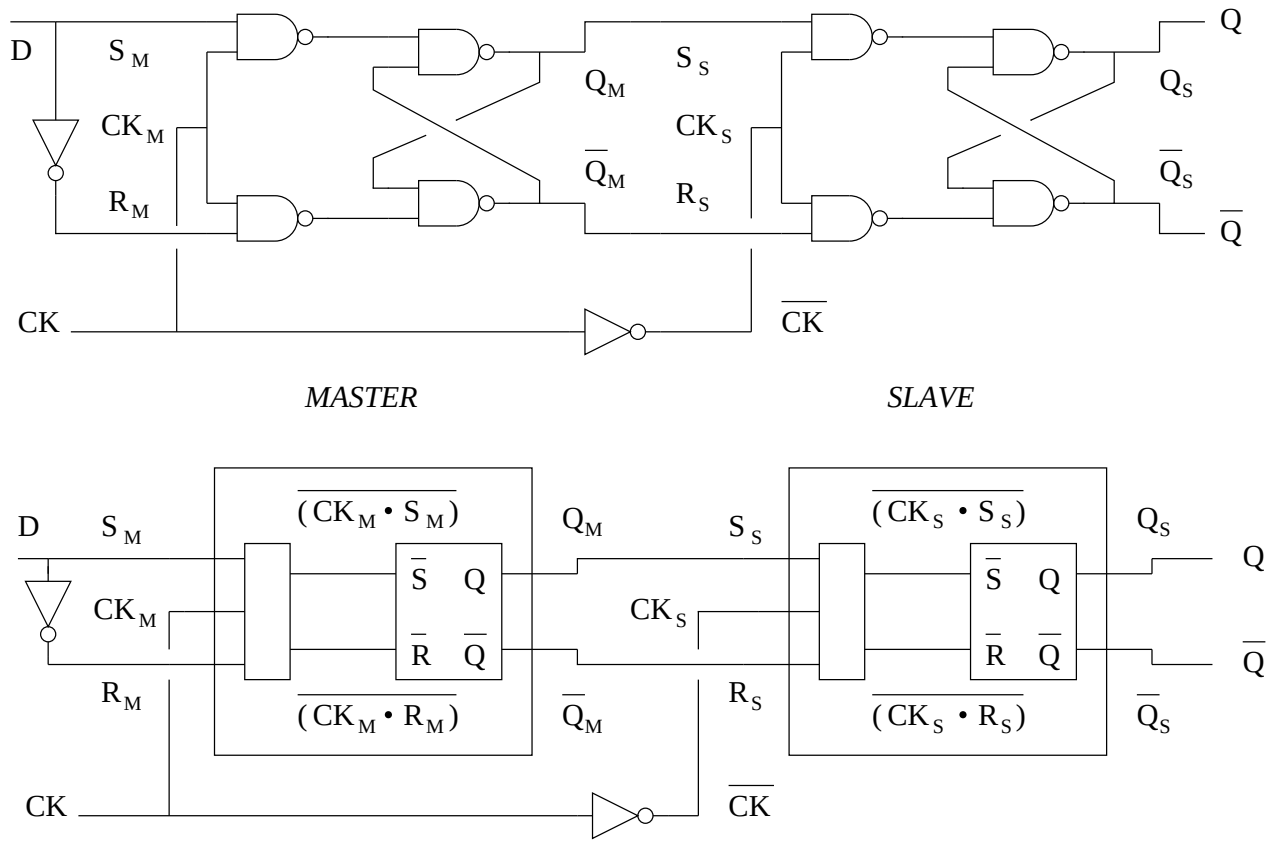


Figura 3.14: Exemplo de implementação de *flip-flop* D do tipo *master-slave*, com base em *flip-flops* SR.

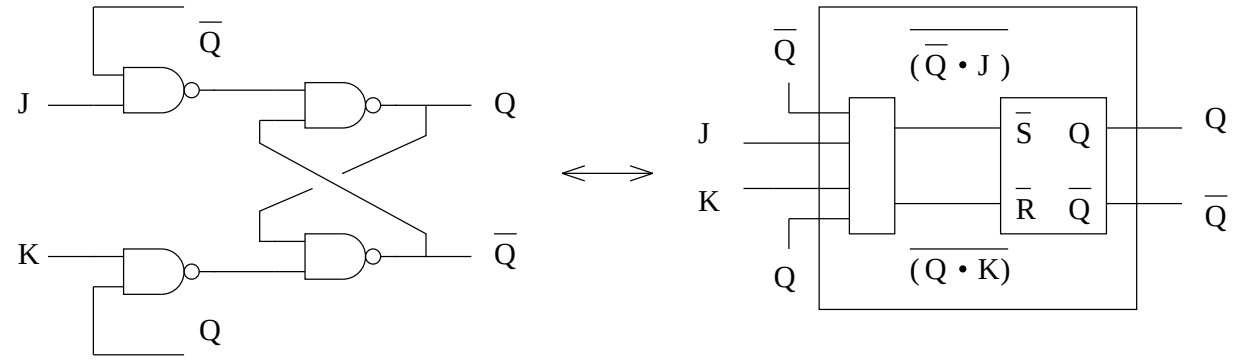


Figura 3.15: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *unclocked*, com problema de oscilação.

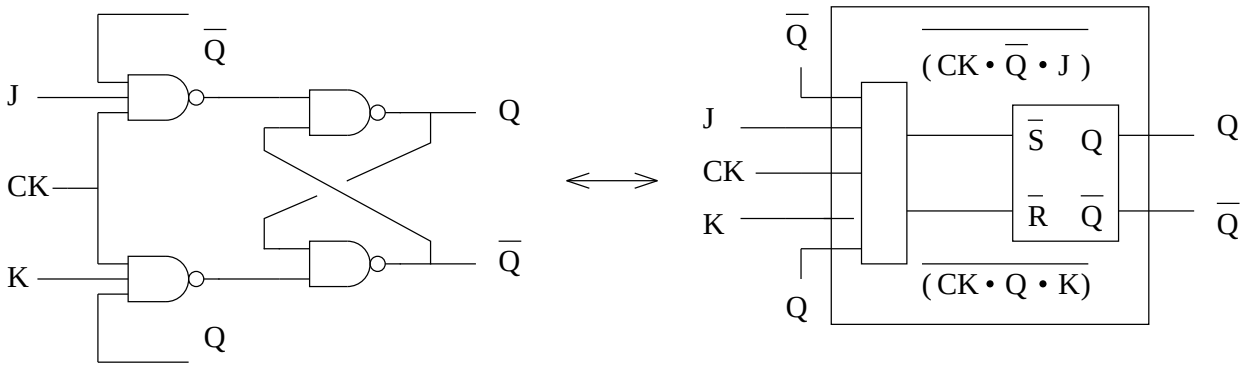


Figura 3.16: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, com problema de oscilação.

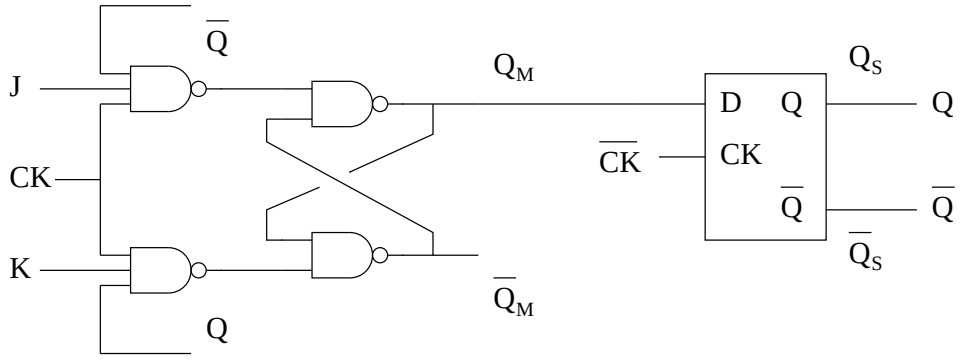


Figura 3.17: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, sem problema de oscilação, devido ao uso de estrutura *master-slave*.

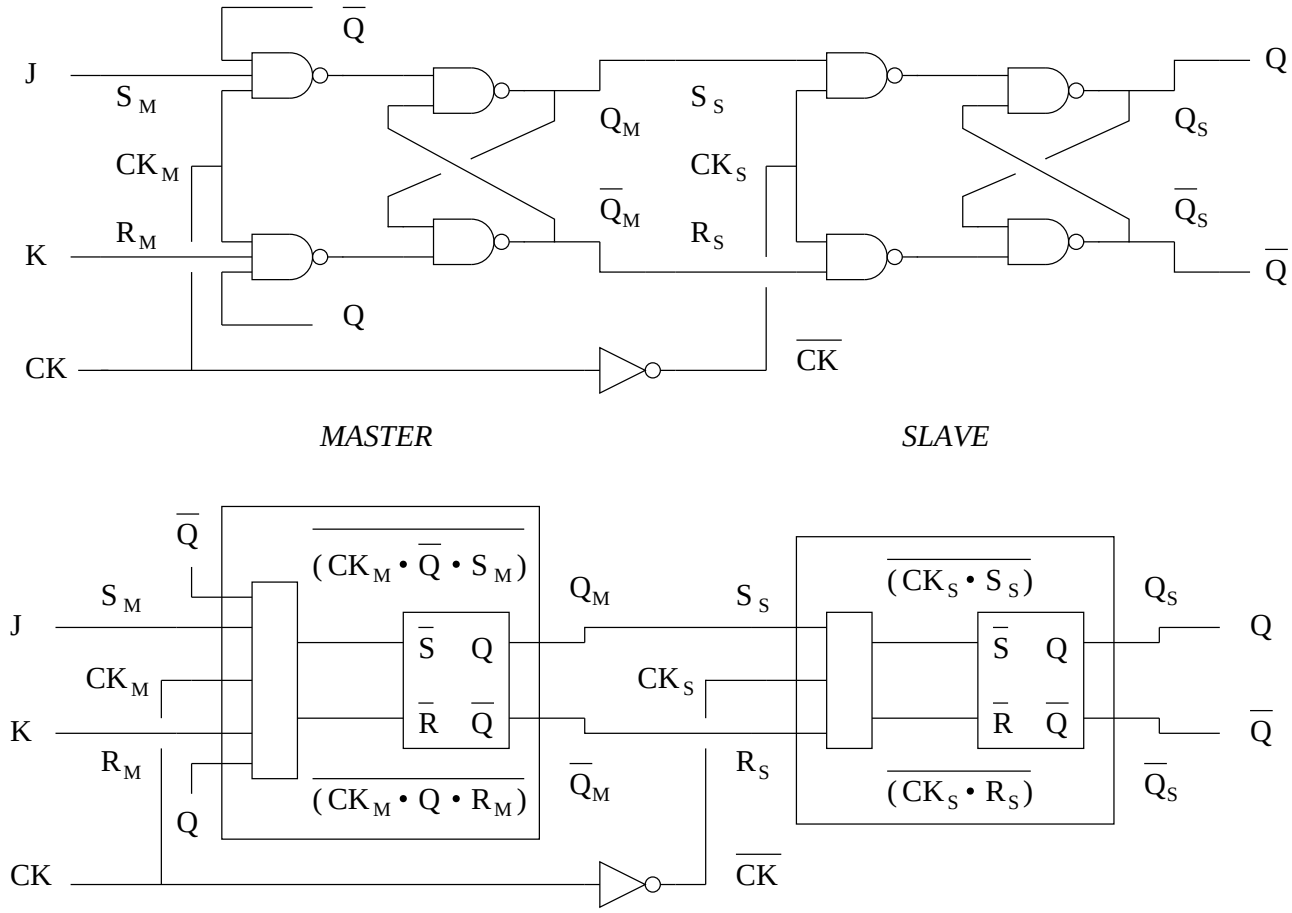


Figura 3.18: Exemplo 1 de implementação de *flip-flop* JK do tipo *master-slave*.

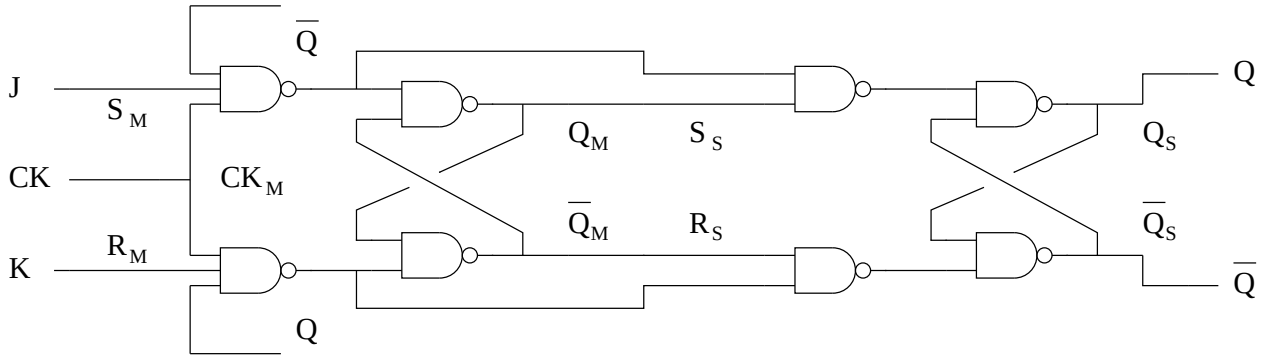


Figura 3.19: Exemplo 2 de implementação de *flip-flop* JK do tipo *master-slave*.

3.11 Variações de funcionalidade

- De acordo com o circuito implementado, um *flip-flop* pode apresentar algumas variações nas suas características funcionais.
- Saídas disponíveis: simples (Q) ou dupla e complementar (Q e $\overline{Q}$).
- Entradas para inicialização da saída: *CLEAR* ($Q = 0$) e *PRESET* ($Q = 1$).
- Tipo de ativação dos sinais de entrada: nível baixo (nível lógico "0") ou nível alto (nível lógico "1").
- Tipo de ativação dos sinais de controle: nível (baixo ou alto), borda (descida ou subida), transição (subida ou descida) ou pulso (negativo ou positivo).

3.12 Diferenças de nomenclatura

- Diversas nomenclaturas diferentes podem ser encontradas na literatura técnica.
- Utilizando como referência os tipos aqui definidos, as nomenclaturas mais comumente encontradas são apresentados na Tabela 3.6.

Nomenclatura	Nomes	Tipos aqui definidos
N1	*Flip-flop*	Todos (os tipos *unclocked* e *clocked* elementar são considerados *flip-flops* elementares)
N2	*Latch*	*unclocked* e *clocked* elementar
	Flip-flop	*clocked master-slave* e *clocked edge-triggered*
N3	*Latch*	*unclocked* e *clocked* elementar
	Latch master-slave	*clocked master-slave*
	Flip-flop	*clocked edge-triggered*
N4	*Latch*	*unclocked*
	Controlled/Clocked-latch	*clocked* elementar
N5	*Positive/Negative-edge flip-flop*	*clocked master-slave*

Tabela 3.6: Diferentes nomenclaturas para *flip-flops*.

Capítulo 4

Circuitos seqüenciais *clock-mode*

4.1 Introdução

- A Figura 4.1 ilustra um modelo genérico para circuitos seqüenciais *clock-mode*.

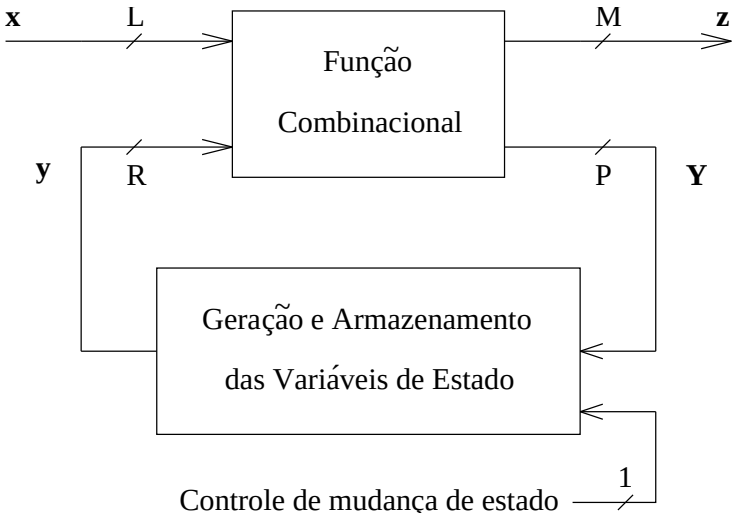


Figura 4.1: Modelo genérico para circuitos seqüenciais *clock-mode*.

- As variáveis de estado são modificadas apenas pela ação de um sinal pulsante, com função de temporização ou de controle, comumente denominado de relógio (*clock*). Apesar de ser um sinal pulsante, não é necessário que o *clock* seja periódico.
- O sinal de *clock* não carrega qualquer tipo de informação. Ele só determina quando haverá mudança de estado.
- As variáveis de excitação, em conjunto com os elementos de armazenamento, determinam qual será a mudança de estado.
- Um *clock* atuando em t_n, com $\boldsymbol{x}^n$, $\boldsymbol{z}^n$, e $\boldsymbol{Y}^n$ estáveis, provoca uma mudança de estado de $\boldsymbol{y}^n$ para $\boldsymbol{y}^{n+1}$.
- O circuito deve estar estável entre dois pulsos de *clock*. Logo, cada circuito possuirá uma freqüência máxima de operação. Tal freqüência será limitada por: i) acionamento dos sinais de entrada, ii) tempos de retardo no bloco "Função Combinacional" e iii) tempos de retardo no bloco "Geração e Armazenamento das Variáveis de Estado".

4.2 Controle de circuitos do tipo *clock-mode*

4.2.1 Características da estrutura *clock-mode*

- $\boldsymbol{z}^n = \boldsymbol{f_1}(\boldsymbol{y}^n, \boldsymbol{x}^n)$, para circuitos do tipo Mealy.
- $\boldsymbol{z}^n = \boldsymbol{f_2}(\boldsymbol{y}^n)$, para circuitos do tipo Moore.
- $\boldsymbol{Y}^n = \boldsymbol{f_3}(\boldsymbol{y}^n, \boldsymbol{x}^n)$.
- $\boldsymbol{y}^{n+1} = \boldsymbol{f_4}(\boldsymbol{Y}^n)$.
- Tempos de propagação:
  - Estabilização da entrada $\boldsymbol{x}$: $\boldsymbol{\Delta t_x}$.
  - Entrada $\boldsymbol{x}$ para saída $\boldsymbol{z}$: $\boldsymbol{\Delta t_{zx}}$.
  - Entrada $\boldsymbol{x}$ para excitação $\boldsymbol{Y}$: $\boldsymbol{\Delta t_{Yx}}$.
  - Excitação $\boldsymbol{Y}$ para estado $\boldsymbol{y}$: $\boldsymbol{\Delta t_{yY}}$.
  - Estado $\boldsymbol{y}$ para saída $\boldsymbol{z}$: $\boldsymbol{\Delta t_{zy}}$.
  - Estado $\boldsymbol{y}$ para excitação $\boldsymbol{Y}$: $\boldsymbol{\Delta t_{Yy}}$.
  - Tempo máximo de propagação: $\Delta t_{max} = max\{\boldsymbol{\Delta t}\} = max\{\Delta t_1, \Delta t_2, \cdots, \Delta t_k\}$.
- Condições de correta operação:
  - Para uma leitura correta dos sinais de saída $\boldsymbol{z}$, os mesmos devem estar estáveis no momento da leitura.
  - Para uma operação previsível do bloco Geração e Armazenamento das Variáveis de Estado (G&A), as variáveis de excitação $\boldsymbol{Y}$ devem estar estáveis no momento do acionamento do bloco.

4.2.2 Controle de circuitos do tipo Moore

- Será assumindo como $\Delta t_x \geq max\{\boldsymbol{\Delta t_x}\}$ o intervalo de tempo entre o acionamento do bloco G&A e a estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que as variáveis de estado $\boldsymbol{y}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yx} \geq max\{\boldsymbol{\Delta t_{Yx}}\}$, a partir da estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que todos os sinais estejam estáveis, as variáveis de estado $\boldsymbol{y}$ estarão estáveis após um tempo $\Delta t_{yY} \geq max\{\boldsymbol{\Delta t_{yY}}\}$, a partir do acionamento do bloco G&A.
- As variáveis de saída $\boldsymbol{z}$ estarão estáveis após um tempo $\Delta t_{zy} \geq max\{\boldsymbol{\Delta t_{zy}}\}$, a partir da estabilização dos sinais de estado $\boldsymbol{y}$.
- Assumindo que os sinais de entrada $\boldsymbol{x}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yy} \geq max\{\boldsymbol{\Delta t_{Yy}}\}$, a partir da estabilização das variáveis de estado $\boldsymbol{y}$.

- Uma vez que, nos circuitos do tipo Moore, a saída depende apenas das variáveis de estado, só é possível ler um valor de saída diferente a cada estado. Assim, ainda que a entrada varie durante o período de tempo de um estado, haverá interesse apenas no seu valor estável final, antes do próximo acionamento que causará uma mudança de estado. Portanto, é necessário considerar apenas o tempo total de estabilização dos sinais de entrada $\boldsymbol{x}$.
- Logo, para cumprir as condições de correta operação, o período de acionamento do bloco G&A deve ser

$$T_{CTRL} = T_{CK} \geq max\{(\Delta t_x + \Delta t_{Yx}), (\Delta t_{yY} + \Delta t_{Yy}), (\Delta t_{yY} + \Delta t_{zy})\} \ . \tag{4.1}$$

- É recomendável que se utilize

$$\Delta t_{yY} < (\Delta t_x + \Delta t_{Yx}) < max\{(\Delta t_{yY} + \Delta t_{Yy}), (\Delta t_{yY} + \Delta t_{zy})\} \ . \tag{4.2}$$

4.2.3 Controle de circuitos do tipo Mealy

- Será assumindo como $\Delta t_x \geq max\{\boldsymbol{\Delta t_x}\}$ o intervalo de tempo entre o acionamento do bloco G&A e a estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que as variáveis de estado $\boldsymbol{y}$ estejam estáveis, as variáveis de saída $\boldsymbol{z}$ estarão estáveis após um tempo $\Delta t_{zx} \geq max\{\boldsymbol{\Delta t_{zx}}\}$, a partir da estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que as variáveis de estado $\boldsymbol{y}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yx} \geq max\{\boldsymbol{\Delta t_{Yx}}\}$, a partir da estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que todos os sinais estejam estáveis, as variáveis de estado $\boldsymbol{y}$ estarão estáveis após um tempo $\Delta t_{yY} \geq max\{\boldsymbol{\Delta t_{yY}}\}$, a partir do acionamento do bloco G&A.
- Assumindo que os sinais de entrada $\boldsymbol{x}$ estejam estáveis, as variáveis de saída $\boldsymbol{z}$ estarão estáveis após um tempo $\Delta t_{zy} \geq max\{\boldsymbol{\Delta t_{zy}}\}$, a partir da estabilização dos sinais de estado $\boldsymbol{y}$.
- Assumindo que os sinais de entrada $\boldsymbol{x}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yy} \geq max\{\boldsymbol{\Delta t_{Yy}}\}$, a partir da estabilização das variáveis de estado $\boldsymbol{y}$.
- Logo, para cumprir as condições de correta operação, supondo uma única mudança nos sinais de entrada a cada estado, o período de acionamento do bloco G&A deve ser

$$T_{CTRL} = T_{CK} \geq max\{(\Delta t_x + \Delta t_{zx}), (\Delta t_x + \Delta t_{Yx}), (\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\} \ . \tag{4.3}$$

- Nesse caso, é recomendável que se utilize

$$\Delta t_{yY} < (\Delta t_x + \Delta t_{Yx}) < max\{(\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\} \tag{4.4}$$

e

$$(\Delta t_x + \Delta t_{zx}) < max\{(\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\} \ . \tag{4.5}$$

4.3 Representação dos estados

- Recursos comuns: texto, equações, tabelas, diagramas gráficos, diagramas temporais.
- Equações: equações de definição dos elementos de memória, equações de próximo estado.
- Tabelas: tabela de transição (de estados), tabela de atribuição de estados e tabela (de transição) de estados.
- Diagramas gráficos: diagrama de fluxo (fluxograma) e diagrama de estados.

4.4 Estado inicial

- Os circuitos seqüenciais, dependendo de sua classe, devem ou podem apresentar um estado explícito de inicialização (*reset state*).
- O estado inicial pode ser um estado extra ou apenas um dos estados já pertencentes à operação normal do circuito.
- Associada ao estado de inicialização, deve haver uma seqüência de inicialização (*reset sequence* ou *synchronizing sequence*).
- Normalmente, a seqüência de inicialização é fornecida por um único e particular sinal de entrada, denominado sinal ou linha de inicialização (*reset line*).
- O sinal de inicialização pode atuar sobre os elementos de memória através das variáveis de excitação ou através de entradas de controle específicas para inicialização (*CLEAR* e *PRESET*), caso existam.

4.5 Classificação quanto à capacidade de memorização

- Circuito com memória não finita
  - Apresenta um estado inicial ou de inicialização (*reset state*).
  - Apresenta um estado final ou um ciclo de estados final.
  - Possui uma seqüência de inicialização (*reset sequence* ou *synchronizing sequence*).
  - Caso particular:
    - Circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$).

- Circuito com memória finita
  - A Figura 4.2 apresenta um circuito com memória finita.
  - Os blocos de retardo unitário $\boldsymbol{D}$ são conjuntos de *flip-flops* do tipo D.
  - Os vetores $\boldsymbol{x}^{n-r}$, $r = 0, 1, \cdots, R$, e $\boldsymbol{z}^{n-s}$, $s = 0, 1, \cdots, S$, representam os sinais de entrada x_i^{n-r}, $i = 1, 2, \cdots, L$, e de saída z_j^{n-s}, $j = 1, 2, \cdots, M$, respectivamente.
  - Neste tipo de circuito: $\boldsymbol{z}^n = f(\boldsymbol{x}^n, \boldsymbol{x}^{n-1}, \cdots, \boldsymbol{x}^{n-R}, \boldsymbol{z}^{n-1}, \cdots, \boldsymbol{z}^{n-S})$.
  - O valor $P = \max\{R, S\}$ é definido como comprimento ou profundidade da memória.
  - Dependendo do projeto, pode haver um estado de inicialização explícito, com uma seqüência de inicialização associada.
  - Um circuito com memória finita pode ser empregado como passo inicial para uma solução com memória não finita.
  - As Figuras 4.3 e 4.4 destacam, respectivamente, dois casos particulares:
    - Circuitos com memória de entrada finita: $\boldsymbol{z}^n = f(\boldsymbol{x}^n, \boldsymbol{x}^{n-1}, \cdots, \boldsymbol{x}^{n-R})$.
    - Circuitos com memória de saída finita: $\boldsymbol{z}^n = g(\boldsymbol{z}^{n-1}, \cdots, \boldsymbol{z}^{n-S})$.

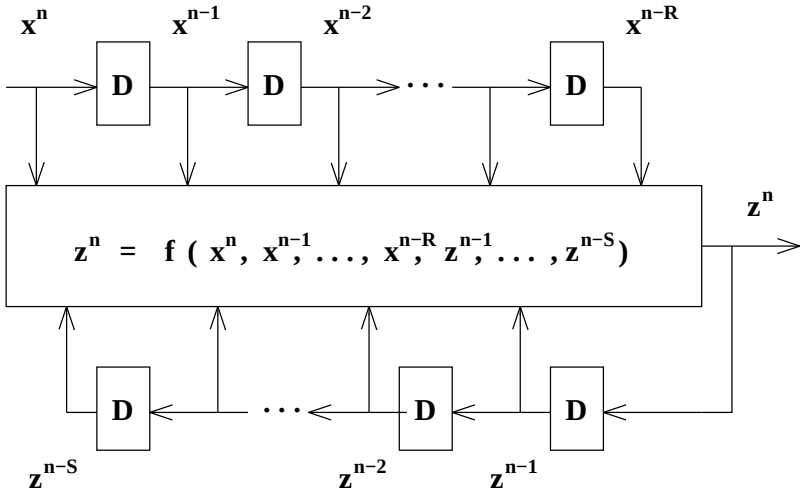


Figura 4.2: Modelo genérico para circuitos com memória finita.

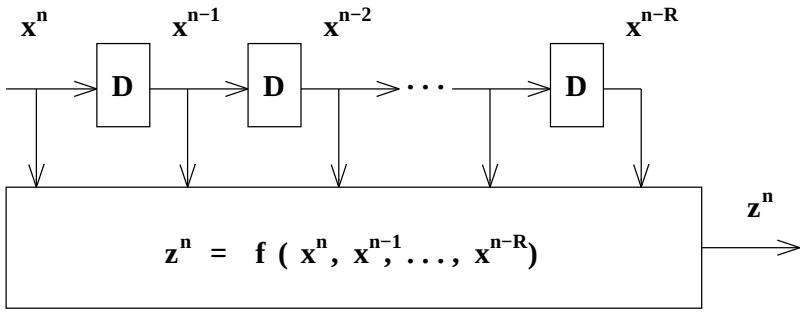


Figura 4.3: Modelo genérico para circuitos com memória de entrada finita.

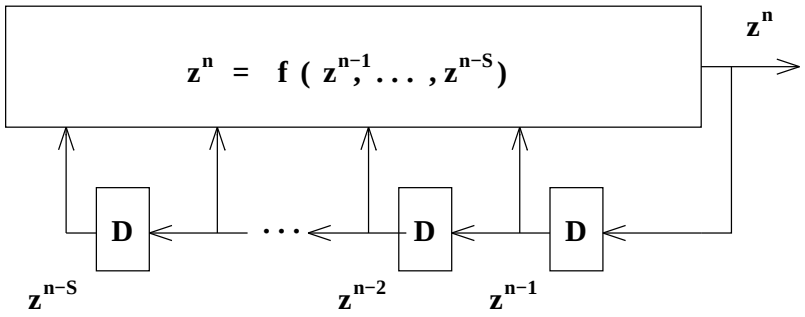


Figura 4.4: Modelo genérico para circuitos com memória de saída finita.

4.6 Análise de circuitos seqüenciais

- Dado um circuito digital seqüencial, existem algumas etapas genéricas para a análise do seu comportamento.
- A seguir, tais etapas são abordadas e alguns exemplos são apresentados.

4.6.1 Etapas de análise

- Passos principais:

  A1 - Circuito a ser analisado.

  A2 - Equações das variáveis de saída, baseadas nas ligações do circuito.

  A3 - Equações das variáveis de excitação, baseadas nas ligações do circuito.

  A4 - Equações de próximo estado, baseadas na operação dos elementos de memória.

  A5 - Tabela de transição de estados (*transition table*), contendo os valores das variáveis de estado.

  A6 - Tabela de atribuição de estados, associando nomes aos valores das variáveis de estado.

  A7 - Tabela de transição de estados (*state table*), contendo os nomes atribuídos aos estados.

  A8 - Diagrama de estados.

4.6.2 Exemplos de análise

- Circuito com memória finita.
- Caso particular de circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$).
- Circuito com memória não finita genérico.

4.7 Projeto de circuitos seqüenciais

- Uma vez que a síntese é o processo reverso em relação à análise, as etapas de projeto podem ser obtidas, a princípio, revertendo-se a ordem das etapas de análise.

- Porém, existe uma profunda diferença entre os dois processos. Na análise, há um único circuito, uma única entrada e um único estado inicial. Portanto, uma saída única é obtida no processo. Por outro lado, no processo de síntese há uma entrada e uma saída, únicas, e se procura por um circuito que realize o mapeamento entrada-saída. A solução nesse caso raramente é única, pois, em cada passo do processo de síntese, decisões podem ser fazer necessárias, gerando uma árvore de opções.

- A seguir, são comentadas as características de projeto para cada um dos tipos de circuito *clock-mode* acima definidos, bem como são especificadas as etapas de projeto para tais circuitos e são apresentados alguns exemplos.

4.7.1 Opções de projeto e suas características

- Circuito com memória finita: ausência de lógica combinacional (ligação por meio de fios) na geração das variáveis de excitação.

- Caso particular de circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$): ausência de lógica combinacional (ligação por meio de fios) na geração das variáveis de saída.

- Circuito com memória não finita genérico: possível existência de lógica combinacional na geração das variáveis de excitação e de saída, a qual pode ser minimizada.

- A Figura 4.5 apresenta os fluxos de projeto para cada uma das três opções.

4.7.2 Etapas de projeto de circuitos seqüenciais

- Os três tipos de projeto abordados possuem etapas que são particulares para cada caso. Porém, pode-se definir um fluxo geral de projeto, que atenda a todos os três tipos. Assim, dependendo do tipo de projeto, pode-se utilizar apenas as etapas necessárias a cada caso.

- Etapas gerais de projeto:

  P1 - Problema a ser resolvido.

  P2 - Descrição funcional do problema (textual).

  P3 - Descrição diagramática, baseada na descrição textual:
  - Diagrama de fluxo (fluxograma).
  - Diagrama de estados.

  P4 - Tabela de transição de estados (*state table*):
  - Diretamente obtida da descrição funcional (circuito com memória finita).
  - Baseada na descrição diagramática (circuito com memória não finita).

P5 - Tentativa de minimização, onde raramente é feita uma minimização global que envolva o circuito combinacional e a memória ao mesmo tempo. Ao invés disso, o mais comum é que se realize o processo em duas etapas:

P5.1 - Memória: Tabela de transição de estados reduzida (*minimal-state table*), baseada em técnicas de minimização de estados.

P5.2 - Combinacional: dependente da classe do circuito a ser projetado. No caso de circuito com memória finita e no caso particular de circuito de Moore, a minimização combinacional é uma característica da estrutura. No caso de circuito com memória não finita, tal minimização é realizada no passo **P6**.

P6 - Tabela de atribuição de estados, baseada em regras genéricas de atribuição.

P7 - Tabela de transição de estados (*transition table*):

- Diretamente da especificação do problema (circuito com memória finita).
- Diretamente da especificação das variáveis de saída (caso particular de circuito de Moore com memória não finita).
- Baseada na atribuição de estados (circuito com memória não finita genérico).

P8 - Escolha dos elementos de memória.

P9 - Equações de entrada dos elementos de memória (variáveis de excitação), baseadas na tabela de transição de estados (*transition table*) e nas tabelas de excitação dos elementos de memória (*excitation table/map* ou *transition list/table/map*).

P10 - Circuito proposto.

P11 - Análise do circuito para verificação de comportamento dos estados não utilizados e não especificados, caso existam.

4.7.3 Exemplos de projeto de circuitos seqüenciais

- Circuito com memória finita de entrada.
- Circuito com memória finita de saída.
- Circuito com memória finita de entrada e de saída.
- Caso particular de circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$).
- Circuito com memória não finita genérico.
- Relacionamento dos três tipos de projeto.

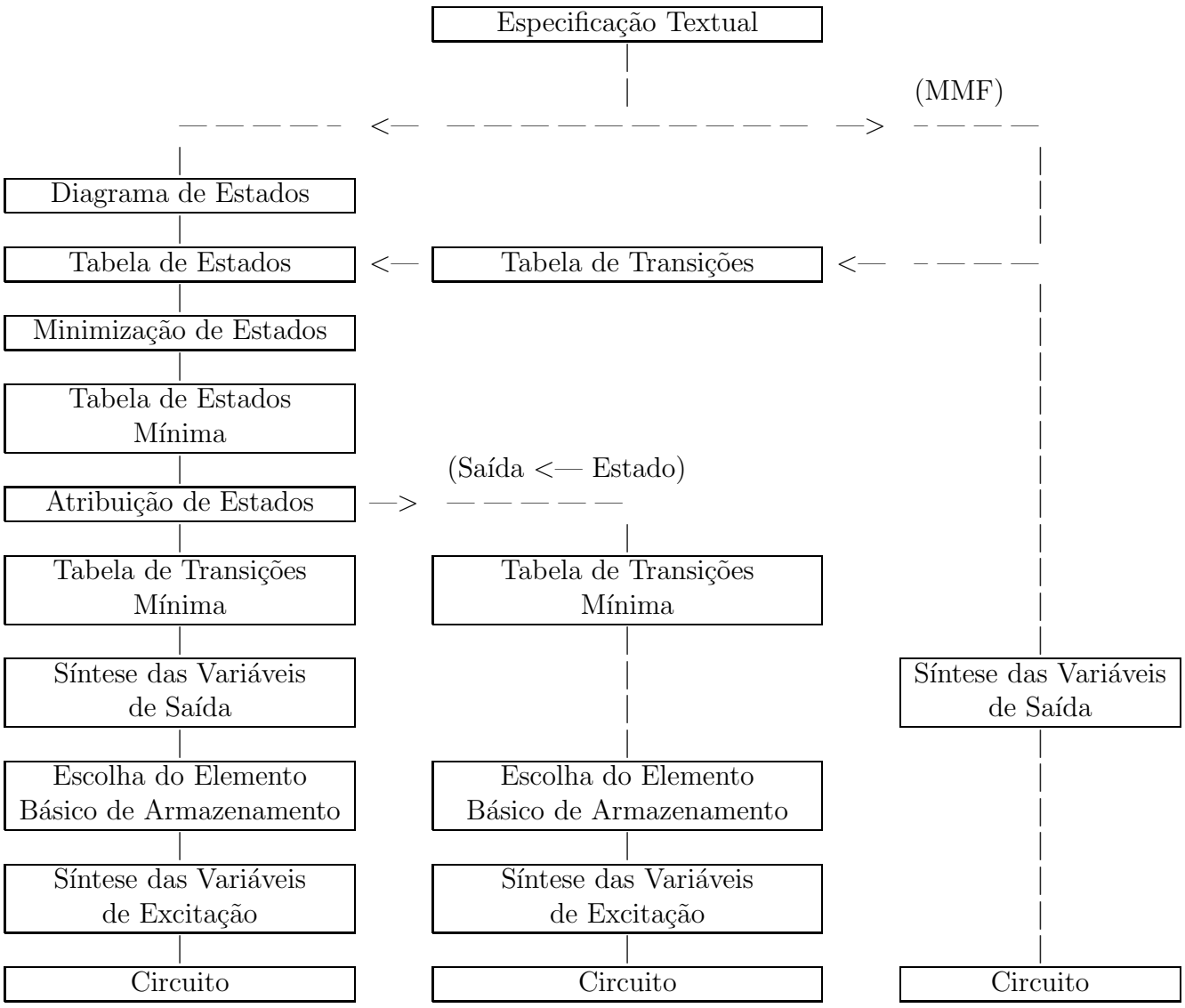


Figura 4.5: Fluxos de projeto para circuitos seqüenciais *clock-mode*: (a) Fluxo genérico, (b) Caso particular de Máquina de Moore e (c) Máquina de Memória Finita (MMF).

4.8 Minimização de estados

4.8.1 Conceitos básicos

- A minimização do número de estados de um circuito seqüencial pode conduzir à redução da quantidade de circuitos lógicos necessários para implementar os estados (bloco Geração e Armazenamento) e as saídas (bloco Função Combinacional).

- Dada uma tabela de transição de estados (*state table*), pode-se constatar que diferentes estados podem realizar a mesma função. Do ponto de vista externo ao circuito, pode-se dizer que não é possível distinguir entre tais estados, uma vez que eles apresentam o mesmo resultado. Nesse caso, tal conjunto de estados pode ser representado por um único estado. Conseqüentemente, a tabela de transição de estados (*state table*) é simplificada e, possivelmente, o circuito lógico minimizado.

- Uma formalismo teórico é apresentado no Apêndice A.

4.8.2 Eliminação de estados redundantes por simples inspeção

- A simples inspeção da tabela de transição de estados (*state table*) pode revelar estados redundantes, os quais podem ser imediatamente unificados em um estado equivalente.

- Em geral, esse método não conduz a um conjunto mínimo de estados, funcionando apenas como um pré-processamento para os demais métodos de minimização.

- Condição de redundância: estados (q^n) que, para cada entrada simples (x^n), conduzem aos mesmos próximos estados e às mesmas saídas (q^{n+1}, z^n), representam um único estado equivalente.

- Algoritmo de eliminação de estados redundantes por simples inspeção:

  EI1 - Verificar a existência de redundância.

  EI2 - Se não houver redundância, ir ao passo EI6.

  EI3 - Se houver redundância, escolher um dos estados redundantes como estado equivalente, mantendo-o na tabela e eliminando todos os demais estados redundantes.

  EI4 - Atualizar a tabela, trocando a designação dos estados eliminados por aquela do estado escolhido como equivalente.

  EI5 - Voltar ao passo EI1.

  EI6 - Fim.

- A Figura 4.6 apresenta um exemplo de eliminação de estados redundantes por simples inspeção.

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$C, 1$	$D, 0$
B	$E, 1$	$F, 0$
C	**D, 0**	**F, 1**
D	$C, 1$	$B, 0$
E	**D, 0**	**F, 1**
F	$E, 1$	$B, 0$

$\longrightarrow$

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$C, 1$	$D, 0$
B	$C, 1$	$F, 0$
C	$D, 0$	$F, 1$
D	**C, 1**	**B, 0**
F	**C, 1**	**B, 0**

$\downarrow$ $\qquad\qquad$ $\downarrow$

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$C, 1$	$D, 0$
C	$D, 0$	$D, 1$
D	$C, 1$	$A, 0$

$\longleftarrow$

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	**C, 1**	**D, 0**
B	**C, 1**	**D, 0**
C	$D, 0$	$D, 1$
D	$C, 1$	$B, 0$

Figura 4.6: Eliminação de estados redundantes através da inspeção da tabela de estados.

4.8.3 Método da partição em classes de estados indistinguíveis (método de Huffman-Mealy)

- O processo é simples, mas não pode ser aplicado para os casos de tabelas de estados não completamente especificadas.
- Ele é baseado no **Teorema 1**, discutido no Apêndice A e apresentado a seguir.
- **Teorema 1**: Suponha-se que os estados de um circuito seqüencial foram particionados em classes disjuntas, onde $p \triangleq q$ denota que os estados p e q pertencem à mesma classe. A partição é composta por classes de equivalência de estados indistinguíveis se e somente se as duas condições seguintes forem satisfeitas por cada par de estados p e q da mesma classe, para cada entrada simples x^n:
  1. $\lambda(p^n, x^n) = \lambda(q^n, x^n)$.
  2. $\delta(p^n, x^n) \triangleq \delta(q^n, x^n)$.
- Conforme definido no Apêndice A, as funções $\lambda(q^n, x^n) = z^n$ e $\delta(q^n, x^n) = q^{n+1}$, representam, respectivamente, a saída atual e o próximo estado.
- Basicamente, o método pode ser dividido em duas partes:
  - Aplicação da condição (1) do **Teorema 1**.
  - Aplicações sucessivas da condição (2) do **Teorema 1**.
- Algoritmo de minimização por partição em classes de estados indistinguíveis:

  HM0 - Tentar eliminar estados redundantes por simples inspeção da tabela de estados original. Se houver alguma eliminação, a tabela de estados reduzida passa a representar a tabela de estados original para o restante do algoritmo. Este passo não é necessário, mas diminui o espaço de busca do algoritmo.

  HM1 - A partir da tabela de estados original, separar, em classes distintas ($C_{z_i} \in C_z$), os estados (e_j) que possuem os mesmos conjuntos de saídas (z_{i_k}), para cada valor da entrada (x_k).

  HM2 - Se houver apenas um estado por classe, ir para o passo HM7.

  HM3 - Se houver pelo menos uma classe atual com mais de um estado, descobrir as classes referentes aos próximos estados de cada estado atual, as quais serão denominadas de próximas classes.

  HM4 - Para cada classe com mais de um estado, verificar as próximas classes, para cada valor da entrada (x).

  HM5 - Se, dentro de uma mesma classe, houver estados com próximas classes diferentes dos demais, separá-los em uma nova classe e retornar para o passo HM2.

  HM6 - Se, dentro de cada classe, não houver estado com próximas classes diferentes dos demais, ir para o passo HM7.

  HM7 - Fim.

- As Figuras 4.7, 4.8 e 4.9 ilustram o processo para diferentes tabelas de estado.

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$C, 1$	$D, 0$
B	$E, 1$	$F, 0$
C	$D, 0$	$F, 1$
D	$C, 1$	$B, 0$
E	$D, 0$	$F, 1$
F	$E, 1$	$B, 0$

$\Longleftrightarrow$

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
q_1	$q_2, 0$	$q_2, 1$
q_2	$q_1, 1$	$q_2, 0$

$\downarrow$ $\uparrow$

Classe(n)	1				2							
Estado(n)	C		E		A		B		D		F	
Classe(n+1)	2	2	2	2	1	2	1	2	1	2	1	2

Figura 4.7: Exemplo de minimização positiva em um passo.

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$B, 0$	$B, 1$
B	$F, 1$	$D, 1$
C	$E, 1$	$G, 1$
D	$A, 0$	$C, 0$
E	$D, 1$	$G, 0$
F	$F, 0$	$A, 0$
G	$C, 1$	$B, 0$

$\downarrow \uparrow$

Classe(n)	0				1		2				3			
Estado(n)	D		F		A		E		G		B		C	
Classe(n+1)	1	3	0	1	3	3	0	2	3	3	0	0	2	2

Figura 4.8: Exemplo de minimização negativa em um passo.

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$C, 0$	$D, 1$
B	$B, 0$	$E, 1$
C	$C, 0$	$E, 1$
D	$E, 1$	$H, 0$
E	$D, 1$	$F, 0$
F	$I, 0$	$C, 1$
G	$J, 0$	$B, 1$
H	$I, 0$	$A, 1$
I	$G, 1$	$E, 0$
J	$H, 1$	$D, 0$

$\Longleftrightarrow$

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
q_0	$q_0, 0$	$q_2, 1$
q_1	$q_3, 0$	$q_1, 1$
q_2	$q_2, 1$	$q_1, 0$
q_3	$q_1, 1$	$q_2, 0$

$\downarrow$

Classe(n)	0												1							
Estado(n)	A		B		C		F		G		H		D		E		I		J	
Classe(n+1)	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	0	1	0	1

$\downarrow$ $\circlearrowleft$

Classe(n)	0						1						2				3			
Estado(n)	A		B		C		F		G		H		D		E		I		J	
Classe(n+1)	0	2	0	2	0	2	3	0	3	0	3	0	2	1	2	1	1	2	1	2

Figura 4.9: Exemplo de minimização positiva em mais de um passo.

4.8.4 Método da tabela de implicação de estados (método de Paul-Unger)

- Processo mais complexo do que o apresentado pelo método da partição em classes.
- Porém, ele é mais genérico, podendo ser aplicado para os casos de tabelas de estados não completamente especificadas.
- **Definição 1**: Um conjunto de estados P é implicado por um conjunto de estados R se, para alguma entrada específica x_k, P é o conjunto de todos os próximos estados $p_i^{n+1} = \delta(r_j^n, x_k^n)$, para todos os estados atuais $r_j \in R$.
- A partir do **Teorema 1** e da **Definição 1**, pode-se dizer que os estados de um conjunto R são equivalentes apenas se todos os estados de um conjunto P, implicado por R, também são equivalentes.
- Para que os estados de um conjunto R sejam equivalentes, todos os pares $(r_i, r_j) \in R$ devem ser equivalentes.
- Logo, para verificar a equivalência dos estados de um conjunto, basta testar a implicação para cada par de estados do conjunto.
- Uma forma de realizar esse teste é montar uma árvore de implicação.
- A partir de um determinado par $(r_i, r_j) \in R$, são determinados os estados implicados para cada entrada. Partindo de cada novo conjunto implicado, a operação é repetida. Se algum conjunto implicado da árvore de (r_i, r_j) não for equivalente, o par inicial (r_i, r_j) não pode ser equivalente.
- Tal processo de investigação, que caracteriza uma prova por absurdo ou contradição, possui uma complexidade muito elevada.
- Uma forma mais eficiente de verificar a equivalência de estados é através de uma prova por negação.
- Nesse caso, os estados são organizados em uma tabela de implicação, onde todas as combinações de pares de estados encontram-se representadas. Para cada par, são determinados os estados implicados, para cada entrada. Em seguida, todos as implicações proibidas são eliminadas da tabela. O processo de proibição é repetido até que nenhuma proibição seja encontrada. Por fim, são listadas as classes de equivalência.
- As proibições iniciais são provenientes de pares de estados que apresentam saídas diferentes para as mesmas entradas.
- Uma tabela de implicação e uma de suas células são apresentadas, respectivamente, nas Figuras 4.10 e 4.11.

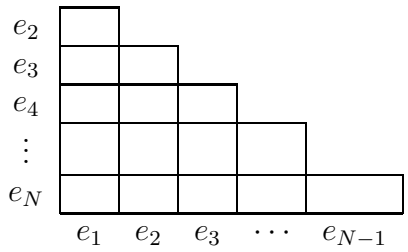


Figura 4.10: Tabela de implicação genérica do método de Paul-Unger.

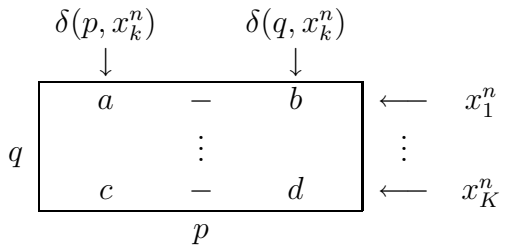


Figura 4.11: Célula genérica da tabela do método de Paul-Unger.

- Algoritmo de minimização por tabela de implicação de estados:

  PU0 - Tentar eliminar estados redundantes por simples inspeção da tabela de estados original. Se houver alguma eliminação, a tabela de estados reduzida passa a representar a tabela de estados original para o restante do algoritmo. Este passo não é necessário, mas diminui o espaço de busca do algoritmo.

  PU1 - A partir da tabela de estados original, separar, em classes distintas ($C_{z_i} \in C_z$), os estados (e_j) que possuem os mesmos conjuntos de saídas (z_{i_k}), para cada valor da entrada (x_k).

  PU2 - Se houver apenas um estado por classe, ir para o passo PU8.

  PU3 - Montar uma matriz triangular inferior, contendo índices horizontais $h_i \equiv e_i \in \{C_z - e_N\}$ e índices verticais $v_j \equiv e_j \in \{C_z - e_1\}$.

  PU4 - Anular todas as posições da matriz, referentes às combinações $h_i \times v_j$, onde $C_z(e_i) \neq C_z(e_j)$.

  PU5 - Preencher todas as posições da matriz, referentes às combinações $h_i^n \times v_j^n$, onde $C_z(e_i^n) = C_z(e_j^n)$, com os pares $(h_i^{n+1} \text{ — } v_j^{n+1})_k$, se $h_i^{n+1} \neq v_j^{n+1}$, para cada valor da entrada (x_k).

  PU6 - Repetir, até que não haja mais anulações, para cada posição não anulada $h_i^n \times v_j^n$ da matriz:

  PU61 - Verificar se os pares $(h_i^{n+1} \text{ — } v_j^{n+1})_k$ foram anulados.

  PU62 - Se, pelo menos um dos pares tiver sido anulado, anular a posição corrente $h_i^n \times v_j^n$ da matriz e notificar a ocorrência de anulação.

  PU7 - Organizar em classes de equivalência os estados cujas combinações $h_i^n \times v_j^n$ não foram anuladas e em classes individuais os demais estados.

  PU8 - Fim.

4.9 Atribuição de estados

4.9.1 Considerações iniciais

- No projeto de um circuito digital seqüencial, a atribuição de estados tem influência direta na síntese da lógica combinacional que gera as variáveis de excitação e as variáveis de saída.
- A prática demonstra que atribuições de estados diferentes podem produzir lógicas combinacionais diferentes.
- Portanto, a fim de se obter o circuito combinacional de menor custo, deve-se procurar a atribuição de estados que favoreça a sua síntese.
- Uma vez que os estados são representados por um conjunto de V variáveis booleanas, duas situações podem ocorrer. Na primeira, o número de estados (S) que se deseja representar é igual ao número de estados representáveis, de forma que $S = 2^V$. Caso contrário, o número de estados a serem representados encontra-se na seguinte faixa: $2^{V-1} < S < 2^V$.
- Quando $S = 2^V$, o problema de atribuição de estados se resume a estabelecer uma relação de equivalência dos estados desejados com as configurações existentes para as variáveis de estado.
- No caso de $2^{V-1} < S < 2^V$, além da equivalência, é necessário também escolher S configurações a serem utilizadas dentre as 2^V existentes.
- Para um número de estados na faixa $2^{V-1} < S \leq 2^V$, pode-se demonstrar que o número total de atribuições (A_{tot}) pode ser calculado por $A_{tot} = \frac{2^V!}{(2^V-S)!}$.
- Porém, muitas dessas atribuições são redundantes, pois representam apenas trocas e/ou complementações lógicas das variáveis de estado.
- Assim, pode-se demonstrar que o número de atribuições efetivamente diferentes (A_{dif}) pode ser calculado por $A_{dif} = \frac{(2^V-1)!}{(2^V-S)!\ V!}$.
- A Tabela 4.1 ilustra algumas possibilidades.

Estados (S)	Variáveis de Estado (V)	Atribuições (A_{dif})
2	1	1
3	2	3
4	2	3
5	3	140
6	3	420
7	3	840
8	3	840
9	4	10.810.800

Tabela 4.1: Número de atribuições de estados efetivamente diferentes.

- Assim sendo, para $S = 3$ ou 4, podem ser realizados 3 projetos, a partir das 3 atribuições possíveis, escolhendo-se o de menor custo.

- Para $S \geq 5$, pode-se visualizar duas soluções:
  - Aplicar um algoritmo que encontre a atribuição de menor custo.
  - Aplicar regras que indiquem um conjunto reduzido de atribuições de menor custo, projetar cada uma delas e realizar a escolha.
- Na literatura relativa ao assunto, podem ser encontradas várias propostas de técnicas a serem aplicadas no processo de atribuição de estados.
- Infelizmente, nenhuma delas apresenta um algoritmo de busca da melhor atribuição.
- Na realidade, são apresentadas regras genéricas, cujo emprego conduz a um conjunto reduzido de atribuições de menor custo.
- Portanto, enfatizando, a função das regras propostas é a de reduzir o número total de atribuições para uma quantidade mínima de atribuições que mereçam ser analisadas.
- De posse de um conjunto reduzido de candidatas a uma atribuição de menor custo, o projetista pode testar as alternativas e realizar a escolha.
- Vale ressaltar, ainda, que a aplicação das regras não garante que a melhor atribuição seja encontrada.
- Dependendo da especificação do circuito seqüencial e do tipo de elemento de memória utilizado, as regras podem apontar para uma solução que não é a de menor custo, porém é bem próxima.

4.9.2 Base teórica para as regras de atribuição de estados

- A atribuição de estados de menor custo é aquela que sintetiza as variáveis de excitação e as variáveis de saída através da menor quantidade de circuito combinacional.
- A redução da quantidade de circuito combinacional empregada é associada à simplificação da equação lógica que o representa.
- Por sua vez, a minimização de uma equação lógica é conseguida através da combinação de mintermos ou maxtermos que possuam adjacência lógica.
- Por adjacência lógica entende-se a situação onde dois mintermos (ou maxtermos) diferem pelo valor de apenas um de seus *bit*.
- No mapa de Karnaugh simbólico da Figura 4.12, os conjuntos de variáveis $\{x_1, x_0\}$ e $\{y_1, y_0\}$ representam, respectivamente, as variáveis de entrada e as variáveis de estado.
- Utilizando-se o mapa na síntese das variáveis de excitação (Y) e das variáveis de saída (z), destacam-se três situações distintas.
- Supondo-se que o mapa se refere à síntese de variáveis de excitação, ocorrerem dois casos que envolvem uma dinâmica de mudança de estados. O primeiro deles é relacionado com a possibilidade de simplificação de valores em linha (v_r). Ele trata da mudança de dois estados atuais para dois próximos estados, para um mesmo valor de entrada. O outro caso é relacionado com a possibilidade de simplificação de valores em coluna (v_c). Ele trata da mudança de um estado atual para dois próximos estados, para dois valores de entrada diferentes.

- Por outro lado, se o mapa se refere à síntese das variáveis de saída, ocorre a terceira alternativa, estática. Nesse caso, os valores atuais ("0" ou "1") da saída podem promover simplificações em linha (v_r) e/ou em colunas (v_c).

- Com base na análise de cada situação, pode-se definir um conjunto de regras básicas que indique uma atribuição de estados adequada.

- Ainda que tais regras não conduzam à maior simplificação possível, elas ajudam a escolher uma solução próxima da ótima.

		y_1y_0			
		00	01	11	10
x_1x_0	00	v_r	v_r		
	01				
	11			v_c	
	10			v_c	

Figura 4.12: Análise de minimização para as equações de excitação e de saída: mapa de Karnaugh simbólico.

Análise para a síntese de variáveis de excitação

- Do mapa de Karnaugh da Figura 4.12, destacam-se duas situações dinâmicas distintas.

- Uma simplificação de linha (v_r) envolve a dinâmica de dois estados atuais para dois próximos estados, considerando uma mesma entrada.

- Por sua vez, uma simplificação de coluna (v_c) envolve a dinâmica de um estado atual para dois próximos estados, considerando duas entradas diferentes.

- Na simplificação de linha (v_r), podem ser identificados alguns subcasos, de acordo com os próximos estados: i) todos iguais para as mesmas entradas, ii) todos iguais para entradas diferentes, iii) alguns iguais para as mesmas entradas, iv) alguns iguais para entradas diferentes, e v) todos diferentes. Tais subcasos não serão analisados, sendo deixados como proposta de exercício.

- A Figura 4.13 apresenta uma tabela de atribuição de estados hipotética. Nesse caso, os estados logicamente adjacentes são: (a, b), (a, c), (b, d) e (c, d).

- As Figuras 4.14 e 4.15 ilustram a análise de minimização para as variáveis de excitação.

- A Figura 4.14 mostra que, se dois estados atuais possuem o mesmo próximo estado e não são logicamente adjacentes, as suas excitações para os elementos de memória (E_{ij}) não poderão ser combinadas. Caso contrário, elas se combinarão com certeza, minimizando a expressão lógica, a menos de uma das variáveis de estado, para a qual não há garantia.

- Assim, desconsiderando-se os subcasos, a recomendação é: "Dois estados que possuam o mesmo próximo estado devem ser logicamente adjacentes!".

- A Figura 4.15 mostra que, se um estado atual possui dois próximos estados que não possuem adjacência lógica, nada garante que as excitações dos elementos de memória (E_{ij} e E_{kl}) serão as mesmas e, portanto, nada garante que elas serão agrupadas para minimizar a expressão lógica. Caso contrário, a minimização é possível com certeza, a menos de uma das variáveis de estado, para a qual não há garantia.

- Nesse caso, a recomendação é: "Dois estados que sejam próximos estados de um mesmo estado devem ser logicamente adjacentes!".

- Uma vez que as duas recomendações envolvem, respectivamente, um impedimento e uma possibilidade, a primeira delas delas deve ser prioritária em relação à segunda.

Estados q	Variáveis de Estado y_1y_0
a	00
b	01
d	11
c	10

Figura 4.13: Análise de minimização para as equações de excitação e de saída: tabela de atribuição de estados hipotética.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	a	$\cdots$	00	$\cdots$
c	10	a	$\cdots$	00	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{00}		E_{10}
	1				

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{10}		E_{00}
	1				

Dinâmica da variável de estado y_0

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	a	$\cdots$	00	$\cdots$
d	11	a	$\cdots$	00	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{00}	E_{10}	
	1				

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{10}	E_{10}	
	1				

Dinâmica da variável de estado y_0

b) Caso com simplificação.

Figura 4.14: Análise de minimização para as equações de excitação: casos de estados atuais com mesmo próximo estado.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
a	00	b	c	01	10
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{00}			
	1	E_{01}			

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{01}			
	1	E_{00}			

Dinâmica da variável de estado y_0

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
a	00	b	d	01	11
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{00}			
	1	E_{01}			

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{01}			
	1	E_{01}			

Dinâmica da variável de estado y_0

b) Caso com simplificação.

Figura 4.15: Análise de minimização para as equações de excitação: casos de estado atual com próximos estados diferentes.

Análise para a síntese de variáveis de saída

- Do mapa de Karnaugh da Figura 4.12, destaca-se uma situação estática, não envolvendo mudanças de estado.
- Com base na atribuição de estados apresentada na Figura 4.13, a Figura 4.16 ilustra a análise de minimização para as variáveis de saída. Se dois estados atuais possuem a mesma saída, para a mesma entrada, e não são logicamente adjacentes, os valores de saída não poderão ser combinados. Caso contrário, os valores de saída serão combinados com certeza, minimizando a expressão lógica.
- Portanto, a recomendação é: "Dois estados atuais que possuam a mesma saída, para a mesma entrada, devem ser logicamente adjacentes!".
- Normalmente, o número de variáveis de saída é menor que o número de variáveis de excitação. Assim sendo, tal recomendação terá a menor prioridade.

q^n	$y_1^n y_0^n$	z_i^n	
		$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	1	0
c	10	1	0
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

x^n \ $y_1^n y_0^n$	00	01	11	10
0		1		1
1		0		0

z_i

Mapa-K da saída z_i

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	z_i^n	
		$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	1	0
d	11	1	0
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

x^n \ $y_1^n y_0^n$	00	01	11	10
0		1	1	
1		0	0	

z_i

Mapa-K da saída z_i

b) Caso com simplificação.

Figura 4.16: Análise de minimização para as equações de saída.

4.9.3 Exemplo de regras simples (Armstrong-Humphrey)

- No projeto de circuitos seqüenciais que possuam um número pequeno de estados, podem ser utilizadas duas regras básicas no processo de atribuição de estados [Arm62], [Hum58].
- Tais regras são originadas na tentativa de minimização da lógica responsável pela geração das variáveis de excitação.
- A principal motivação para o emprego destas regras é que elas são de curta descrição, de fácil compreensão, de simples aplicação e conduzem a bons resultados.
- Regras:
  - **Regra 1**: Dois ou mais estados que possuam o mesmo próximo estado devem ser logicamente adjacentes.
  - **Regra 2**: Dois ou mais estados que sejam próximos estados de um mesmo estado devem ser logicamente adjacentes.
- É importante ressaltar que as regras são listadas em ordem decrescente de prioridade.
- A Figura 4.17 ilustra as regras descritas acima.

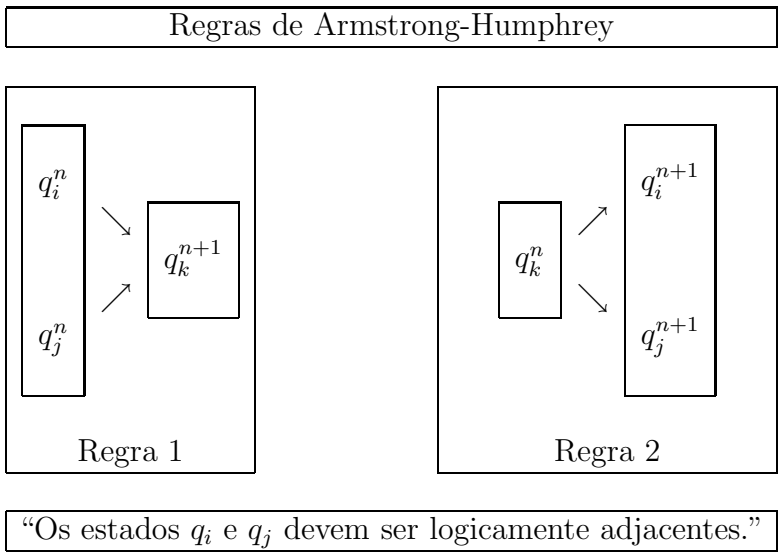


Figura 4.17: Ilustração das regras de Armstrong-Humphrey.

4.9.4 Exemplo de regras mais refinadas

- Um conjunto de regras mais completo pode ser obtido: i) ao se detalhar a Regra 1, anteriormente apresentada, e ii) ao se incorporar a tentativa de minimização da lógica responsável pela geração das variáveis de saída.
- É importante ressaltar que as regras são listadas em ordem decrescente de prioridade.
- Regras:
  - **Regras 1**:
    - **Regra 1a**: Os estados que possuam todos os próximos estados iguais, coluna a coluna, devem ser logicamente adjacentes. Se possível, os próximos estados também devem ser logicamente adjacentes, de acordo com a **Regra 2**.
    - **Regra 1b**: Os estados que possuam todos os próximos estados iguais, mas em colunas diferentes, devem ser logicamente adjacentes se os próximos estados também puderem ser logicamente adjacentes.
    - **Regra 1c**: Os estados que possuam alguns do próximos estados iguais devem ser logicamente adjacentes. A prioridade de adjacência será maior para os estados que apresentarem um maior número de próximos estados iguais.
  - **Regra 2**: Os próximos estados provenientes de um mesmo estado atual devem ser logicamente adjacentes.
  - **Regra 3**: As atribuições devem ser feitas de forma a simplificar os mapas das variáveis de saída. Assim sendo, os estados que possuam as mesmas saídas, para as mesmas entradas, devem ser logicamente adjacentes.

4.10 Efeitos causados por estados extras

4.10.1 Definição do problema

- No projeto de circuitos seqüenciais, é comum ocorrer a situação onde o número total de estados que pode ser implementado pelo circuito é maior do que o número total de estados que constam na sua especificação.
- Uma vez que, teoricamente, não haverá transições dos estados principais para os estados extras, os valores de próximo estado e de saída para os estados extras podem ser assumidos como não especificados (*don't care* ou com valor lógico "X").
- Tal decisão de projeto acarreta duas conseqüências imediatas. Por um lado, evita-se empregar uma quantidade extra de circuito lógico combinacional, responsável pelo correto funcionamento a partir dos estados extras. Além disso, os valores lógicos "X" podem acarretar simplificações no projeto do circuito lógico principal, durante a síntese das variáveis de excitação.
- Na prática, porém, algum mal funcionamento do circuito pode colocá-lo em um dos estados extras.
- Por essa razão, deve-se realizar uma análise do circuito projetado, de modo a verificar o comportamento de tais estados.

- As seguintes situações podem ocorrer nos circuitos cujo projeto contém estados não especificados:
  - No caso particular do uso de *flip-flops* do tipo SR, pode acontecer alguma indeterminação no circuito seqüencial devido a indeterminações nos *flip-flops* ($S = R = 1$).
  - As saídas do circuito podem apresentar valores não esperados e/ou não especificados.
  - Podem surgir estados extras isolados (*dead states*) ou ciclos isolados de estados extras (*dead cycles*), totalmente independentes dos estados relativos à operação normal do circuito seqüencial projetado.
  - Todos os estados extras podem formar seqüências de estados que convergem para os estados relativos à operação normal do circuito seqüencial projetado. O diagrama de estado de tais circuitos é chamado de arbusto (*bush*), sendo o conjunto dos estados normais de operação denominado de tronco (*trunk*) e as seqüências de estados extras de ramos (*branches*). Nesses casos, o circuito seqüencial é dito auto-corretivo (*self-correcting*).

4.10.2 Possíveis soluções

- As soluções para o retorno do circuito aos seus estados principais, a partir de algum estado extra, podem envolver dois tipos de ações.
- Adotando-se uma correção ativa, pode-se empregar circuitos lógicos adicionais, com a função de auxiliar na detecção dos estados extras e na atuação sobre o circuito.
- Em um tipo de correção passiva, pode-se projetar o circuito de tal forma que seu Diagrama de Estados final seja um arbusto.
- Ações ativas (após o projeto):
  - Detecção de erro: que exige um circuito adicional para identificação de um estado extra.
  - Sinalização de erro: que pode ser implementada através de um sinal extra de saída (*flag* de erro) ou de um valor de saída não especificado.
  - Interrupção do sinal de *clock*: que necessita de um circuito extra para mascaramento do sinal de *clock* original.
  - Correção ativa: que executa o retorno a um dos estados principais através de um sinal de RESET.
- Ações passivas (durante o projeto):
  - Verificar os mapas-K de excitação, para evitar que ocorram indeterminações (valores $S = R = 1$) em *flip-flops* do tipo SR.
  - Verificar os mapas-K de excitação, para evitar que ocorram *dead states* e/ou *dead cycles*.
  - Verificar os mapas-K de saída, para garantir consistência nos valores das mesmas.
- Deve ser ressaltado que, em projetos onde a operação correta é fundamental, ambos os tipos de ações devem ser empregados.

Capítulo 5

Circuitos seqüenciais *pulsed*

5.1 Introdução

- A Figura 5.1 apresenta um modelo genérico para circuitos seqüenciais *pulsed*.

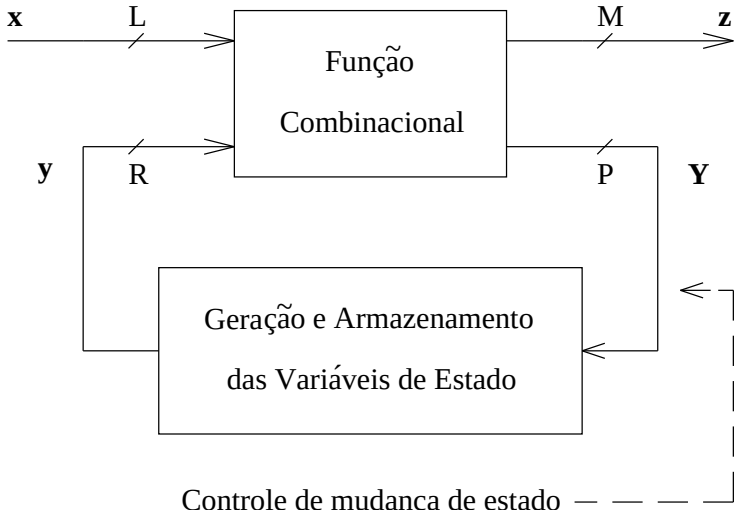


Figura 5.1: Modelo genérico para circuitos seqüenciais *pulsed*.

- O modelo destaca a ausência de um sinal especial de relógio ou *clock*, que atue diretamente sobre o circuito de memória, destinado puramente ao sincronismo.

- Uma mudança de estado é provocada pela ocorrência de um pulso em um dos sinais de entrada.

- Qualquer um dos sinais de entrada pode ser do tipo pulso.

- Os sinais de entrada x_i podem ser tanto do tipo nível quanto do tipo pulso. Porém, é obrigatório que pelo menos um deles seja do tipo pulso.

- No caso de circuitos seqüenciais do tipo Mealy, por definição, as saídas poderão ser do tipo nível e/ou do tipo pulso, uma vez que poderão ser provenientes das combinações dos níveis das variáveis de estado com os níveis e os pulsos dos sinais de entrada. Porém, o mais comum é que as saídas sejam todas do tipo pulso.

- Para os circuitos seqüenciais do tipo Moore, as saídas deverão ser do tipo nível e deverão permanecer estáveis durante o intervalo de tempo entre dois pulsos de entrada consecutivos.

- Os elementos de memória podem ser dos tipos *unclocked* ou *clocked*.

- Em relação às combinações de sinais dos tipos nível e pulso, vale destacar os seguintes aspectos:

  - Uma vez que combinações lógicas AND e OR entre pulsos positivos e negativos produzem resultados indeterminados, apenas um tipo de pulso (positivo ou negativo) deve ser usado.
  - Após a escolha do tipo de pulso a ser utilizado (positivo ou negativo), ainda deve ser lembrado que algumas interações entre sinais dos tipos nível e pulso geram resultados indeterminados para as operações lógicas AND e OR.
  - Portanto, considerando-se que os sinais x_l, x_p e $\overline{x}_p$ representam, respectivamente, sinais dos tipos nível, pulso positivo e pulso negativo, as seguintes combinações podem ser empregadas:

$$* \begin{cases} x_l \cdot x_l = x_l \quad , \quad x_l + x_l = x_l \quad , \quad x_l \cdot x_p = x_p \quad , \quad x_p + x_p = x_p \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{ou} \\ x_l \cdot x_l = x_l \quad , \quad x_l + x_l = x_l \quad , \quad x_l + \overline{x}_p = \overline{x}_p \quad , \quad \overline{x}_p \cdot \overline{x}_p = \overline{x}_p \; . \end{cases}$$

5.2 Restrições de operação

- Os circuitos seqüenciais *pulsed* apresentam as seguintes restrições para o seu correto funcionamento:

  - Deve ser garantido que os elementos de memória operem de tal forma que ocorra apenas uma mudança de estado para cada pulso de entrada.
  - Todos os pulsos de entrada devem apresentar uma duração (largura de pulso) suficiente para o correto acionamento dos elementos de memória.
  - As bordas de disparo dos pulsos de entrada consecutivos, em um mesmo sinal de entrada ou em sinais de entrada diferentes, devem ser espaçadas de um intervalo maior que o tempo de mudança de estado dos elementos de memória.
    - Como conseqüência desta restrição, é vetada a ocorrência de pulsos simultâneos em sinais de entrada diferentes.
  - As entradas do tipo nível devem estar estáveis quando ocorrer um pulso em qualquer das entradas do tipo pulso.

5.3 Classificação quanto aos pulsos de entrada

- Três classes de circuitos seqüenciais do tipo *pulsed* podem ser destacadas: *controlled-clock*, *pulse-mode* e *ripple-clock*.
- Circuitos do tipo *controlled-clock* são casos particulares, sujeitos a mais restrições. Ainda assim, tal abordagem permite o projeto de sistemas digitais mais complexos do que aqueles pertencentes à classe de circuitos *clock-mode*.
- Circuitos do tipo *pulse-mode* representam uma classe mais geral dentro dos circuitos do tipo *pulsed*. Eles podem ser empregados nos casos onde as restrições de sincronismo dos circuitos *clock-mode* e *controlled-clock* não possam ser cumpridas.
- Circuitos do tipo *ripple-clock* resultam de um tentativa de otimização que pode levar à redução da quantidade de *hardware* em detrimento da freqüência máxima de operação.
- Circuitos do tipo *controlled-clock*:
  - Os elementos de memória são do tipo *clocked*.
  - Existe somente uma entrada pulsada, sendo esta periódica e denominada de *clock*.
  - O sinal de *clock* não é aplicado diretamente nas entradas de controle dos elementos de memória. Ele é combinado com os outros sinais de entrada e/ou com as variáveis de estado para gerar fontes secundárias de sinais pulsados, sincronizados com o sinal de *clock*.
- Circuitos do tipo *pulse-mode*:
  - Os elementos de memória podem ser dos tipos *unclocked* ou *clocked*.
  - Normalmente, existe mais de uma entrada pulsada.
  - Podem ser destacados dois casos: i) coexistência de entradas dos tipos nível e pulso e ii) existência apenas de entradas do tipo pulso.
  - Geralmente, os diversos sinais de entrada pulsantes são aperiódicos e temporalmente descorrelacionados.
- Circuitos do tipo *ripple-clock*:
  - Existe, pelo menos, uma entrada pulsada.
  - Existe, pelo menos, um elemento de memória ativado pelos pulsos de entrada. Em seguida, as saídas desse elemento servem de sinal de ativação para outros elementos de memória, e assim consecutivamente, até que todos os elementos de memória tenham sido ativados.
  - As entradas pulsadas podem ser periódicas ou não.
  - O intervalo de tempo entre pulsos de entrada consecutivos deve levar em conta o tempo de propagação de disparos sucessivos dos elementos de memória. Isso pode ser controlado através do pior caso ou através de sinais de término de disparos.
  - Geralmente, o circuito apresenta estados intermediários não estáveis (transitórios). Se necessário for, as saídas devem ser controladas pelos pulsos de entrada, a fim de que apresentem apenas os resultados estáveis.

5.4 Circuitos *pulse-mode*

5.4.1 Motivação

- Existem situações onde as restrições de sincronismo para os circuitos *clock-mode* e *controlled-clock* não podem ser atendidas.
- Uma situação típica é a interface entre subsistemas projetados independentemente uns dos outros.
- Outra situação típica é a interconexão de subsistemas implementados com famílias lógicas diferentes, onde a diferença de taxa de operação é significativa.
- Por exemplo, um sinal de saída do tipo nível em um subsistema com taxas elevadas de chaveamento pode ser interpretado com um pulso de entrada em um subsistema mais lento.
- Utilizando a técnica de projeto *pulse-mode*, o projetista ganha liberdade para designar quais sinais serão interpretados como sendo do tipo nível ou do tipo pulso.

5.4.2 Mudanças nas representações

- O diagrama de estados, a tabela de transição de estados (*state table*) e o mapa-K usados na síntese de circuitos *pulse-mode* apresentam algumas mudanças em relação àqueles que são empregados em circuitos *clock-mode*.
- Tanto a sintaxe quanto a semântica de tais representações sofrem modificações.
- Diversas sintaxes, bem como seus significados, podem ser propostas.
- A sintaxe e a semântica utlizadas no presente texto são detalhadas a seguir.
- Assim como nos circuitos *clock-mode*, os valores $x_l = 0$ e $x_l = 1$, de uma entrada do tipo nível, e os valores $z_l = 0$ e $z_l = 1$, de uma saída do tipo nível, representam os níveis lógicos que tais sinais podem assumir.
- No diagrama de estados, a ausência ou a presença de um pulso (positivo ou negativo) em um sinal de entrada pulsante x_p é representada, respectivamente, pela ausência ou pela presença da variável x_p (pulso positivo) ou de sua negação lógica $\overline{x}_p$ (pulso negativo).
- No diagrama de estados, a ausência ou a presença de um pulso (positivo ou negativo) em um sinal de saída pulsante z_p é representada, respectivamente, pelo valor lógico "0" ou pela presença da variável z_p (pulso positivo) ou de sua negação lógica $\overline{z}_p$ (pulso negativo).
- Na tabela de estados, os valores $z_p = \overline{z}_p = 0$ e $z_p = \overline{z}_p = 1$, de saídas pulsantes z_p (pulso positivo) e $\overline{z}_p$ (pulso negativo), representam, respectivamente, a ausência e a presença de um pulso em z_p e $\overline{z}_p$.
- Entradas não especificadas nas transições do diagrama de estados, bem como as saídas nesses casos, são representadas na tabela de estados como *don't care* ("X").
- No diagrama de estados, a especificação conjunta de duas ou mais variáveis de entrada do tipo pulso, (x_{p1},x_{p2}, $\cdots$), indica apenas que a ocorrência de um pulso em qualquer dos sinais x_{pi} acarretará uma mudança de estado. Afinal, deve ser lembrado que, devido às restrições de operação, é proibida a ocorrência de pulsos simultâneos.

- Como conseqüência das possibilidades de combinação entre sinais do tipo nível e sinais do tipo pulso, as variáveis de saída e as variáveis de excitação devem ser geradas por SOP envolvendo pulsos positivos ou por POS envolvendo pulsos negativos.

- A sintaxe e a semântica do mapa-K, usado na síntese das funções combinacionais, vão depender do tipo de elemento de memória utilizado.

- Na síntese das variáveis pulsadas (excitação ou saída), é comum que se utilize os valores "0" e "1" para representar, respectivamente, a ausência ou a presença de pulsos. Esse tipo de representação é mais adequado para um tratamento por computador. Para uso humano, pode ser de grande auxílio utilizar um sinal indicativo de pulso ("Π"), conforme ilustrado na Figura 5.2.

- Vale a pena ressaltar que, por vezes, o funcionamento desejado do circuito produz um diagrama e uma tabela de estados não completamente especificados. Nesses casos, cabe ao projetista decidir como proceder em relação aos itens não especificados durante a realização do projeto.

- A Figura 5.3 apresenta exemplos de tabelas de estados para circuitos *pulse-mode* Mealy e Moore. A tabela da Figura 5.3.a especifica que deverá ocorrer um pulso na saída z_p quando o circuito estiver no estado $q = B$ e ocorrer um pulso na entrada x_{p2} ou quando o circuito estiver no estado $q = C$ e ocorrer um pulso na entrada x_{p1}. Por sua vez, a tabela da Figura 5.3.b determina que a saída deverá assumir o nível $z_l = 1$ enquanto o circuito estiver no estado $q = D$ e não ocorrer um pulso em qualquer das entradas.

y_1y_2 \ $x_{p1}x_{p2}$	00	01	11	10
00	0	$0/1/X$	—	$0/1/X$
01	0	$0/1/X$	—	$0/1/X$
11	0	$0/1/X$	—	$0/1/X$
10	0	$0/1/X$	—	$0/1/X$

$\longleftrightarrow$

y_1y_2 \ $x_{p1}x_{p2}$	00	01	11	10
00	0	$0/\Pi/X$	—	$0/\Pi/X$
01	0	$0/\Pi/X$	—	$0/\Pi/X$
11	0	$0/\Pi/X$	—	$0/\Pi/X$
10	0	$0/\Pi/X$	—	$0/\Pi/X$

Figura 5.2: Equivalência de notações para mapa de Karnaugh utilizado na síntese de variáveis pulsadas.

q^n	q^{n+1}, z_p	
	x_{p1}	x_{p2}
A	$A, 0$	$B, 0$
B	$-$, $-$	$C, 1$
C	$A, 1$	$D, 0$
D	$A, 0$	$A, 0$

a) Circuito do tipo Mealy.

q^n	q^{n+1}		z_l^n
	x_{p1}	x_{p2}	
A	B	–	0
B	D	C	0
C	A	A	0
D	C	A	1

b) Circuito do tipo Moore.

Figura 5.3: Tabelas de estados para circuitos *pulse-mode* Mealy e Moore.

5.4.3 Exemplos de projeto

- Exemplo utilizando *flip-flop* JK, *master-slave*, ativado por pulso nas entradas J e K, enquanto a entrada de controle de sincronismo C é mantida em nível lógico "1".

y_1y_2 \ $x_{p1}x_{p2}$	00	01	11	10
00	0	0/Π	—	0/Π
01	0	0/Π	—	0/Π
11	0	0/Π	—	0/Π
10	0	0/Π	—	0/Π

(*a*)

y_1y_2	x_{p1}	x_{p2}
00	0/Π	0/Π
01	0/Π	0/Π
11	0/Π	0/Π
10	0/Π	0/Π

(*b*)

Figura 5.4: Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se duas entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado.

y_1y_2 \ $x_{p1}x_{p2}x_{p3}$	000	001	011	010	100	101	111	110
00	0	0/Π	—	0/Π	0/Π	—	—	—
01	0	0/Π	—	0/Π	0/Π	—	—	—
11	0	0/Π	—	0/Π	0/Π	—	—	—
10	0	0/Π	—	0/Π	0/Π	—	—	—

(*a*)

y_1y_2	x_{p1}	x_{p2}	x_{p3}
00	0/Π	0/Π	0/Π
01	0/Π	0/Π	0/Π
11	0/Π	0/Π	0/Π
10	0/Π	0/Π	0/Π

(*b*)

Figura 5.5: Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se três entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado.

5.5 Circuitos *ripple-clock*

5.5.1 Motivação

- A classe de circuitos *ripple-clock* surge como uma tentativa de otimização no acionamento dos elementos de memória do circuito seqüencial.
- A mudança na forma de acionamento dos elementos de memória pode levar a uma simplificação da lógica combinacional do circuito seqüencial.
- Tal simplificação acarreta uma redução da quantidade de *hardware* do circuito combinacional.

5.5.2 Operação

- Nos circuitos do tipo *clock-mode*, os elementos de memória são acionados simultaneamente pelo sinal de sincronismo (*clock*).
- De forma semelhante, nos circuitos do tipo *pulse-mode*, os elementos de memória são potencialmente acionados em paralelo. A diferença, neste caso, é que, dependendo dos sinais de entrada, alguns elementos de memória podem não ser acionados em uma determinada mudança de estado. Ainda assim, a forma de acionamento é estruturalmente paralela.
- Nos circuitos *ripple-clock*, o acionamento é realizado por uma seqüência de eventos. Um sinal de entrada provoca o acionamento de um ou mais elementos de memória. Por sua vez, as modificações nas saídas destes elementos acionam outros elementos de memória. Este mecanismo se repete até que um último conjunto de elementos de memória seja ativado, completando a mudança de estado do circuito seqüencial.

5.5.3 Desvantagens

- As desvantagens deste tipo de acionamento são: i) o aumento do tempo de estabilização nas mudancas de estado, o que é equivalente à redução da freqüência máxima de operação do circuito seqüencial e ii) o surgimento de estados e de conjunto de saídas intermediários (instáveis) durante uma mudança de estados estáveis.
- No cálculo do período mínimo para o sinal de acionamento inicial, deve-se levar em conta o pior caso, que é quando ocorrem todos os níveis de acionamento intermediários.

5.5.4 Técnica de projeto

- Na síntese da lógica combinacional para os circuitos *clock-mode* (ou *pulse-mode*), torna-se necessário que os valores das variáveis de excitação que preenchem os mapas-K sejam rigidamente controlados, pois os elementos de memória serão constantemente (ou potencialmente) acionados, independentemente do estado em que se encontre o circuito.
- No caso dos circuitos *ripple-clock*, os elementos de memória poderão ser acionados apenas quando necessário. Portanto, para os estados onde não ocorrerá acionamento, os valores da variáveis de excitação podem ser considerados *don't care* ("X"), o que pode conduzir a simplificações na lógica combinacional.
- O desafio, portanto, é obter um arranjo de acionamentos que reduza ao máximo a lógica combinacional necessária.

5.5.5 Exemplo

- O exemplo mais clássico é a obtenção do circuito *ripple-clock* para um contador binário, a partir de um projeto de circuito *clock-mode* que utiliza um *flip-flop* JK sensível a transição.

5.6 Circuitos *controlled-clock*

- Os elementos de memória são do tipo *clocked*.
- Assim como nos circuitos seqüenciais *clock-mode*, existe somente uma entrada pulsada, sendo esta periódica e denominada de *clock*.
- Porém, o sinal de *clock* não é aplicado diretamente nas entradas de controle dos elementos de memória.
- Como o próprio nome indica, o sinal de *clock* principal (*master clock*) é combinado com sinais de controle do tipo nível (sinais de entrada e/ou variáveis de estado) para gerar fontes secundárias de sinais pulsados, sincronizados com o sinal de *clock*.
- Tais sinais pulsados secundários são aplicados nas entradas de controle dos elementos de memória ou ainda enviados para circuitos do tipo *pulse-mode*.
- As Figuras 5.6 e 5.7 apresentam exemplos de controle de sinal de *clock*.

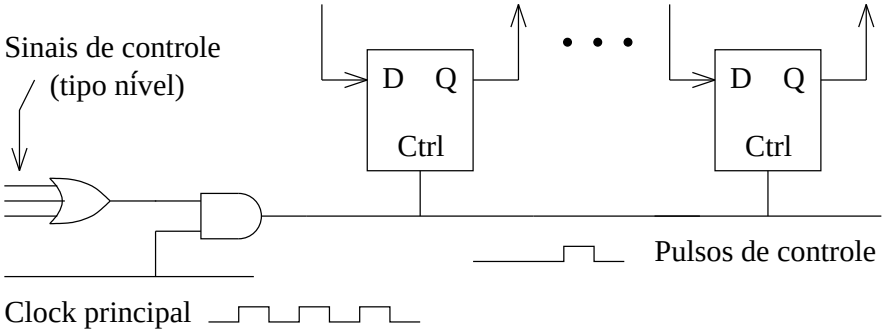


Figura 5.6: Exemplo 1 de controle de sinal de *clock*.

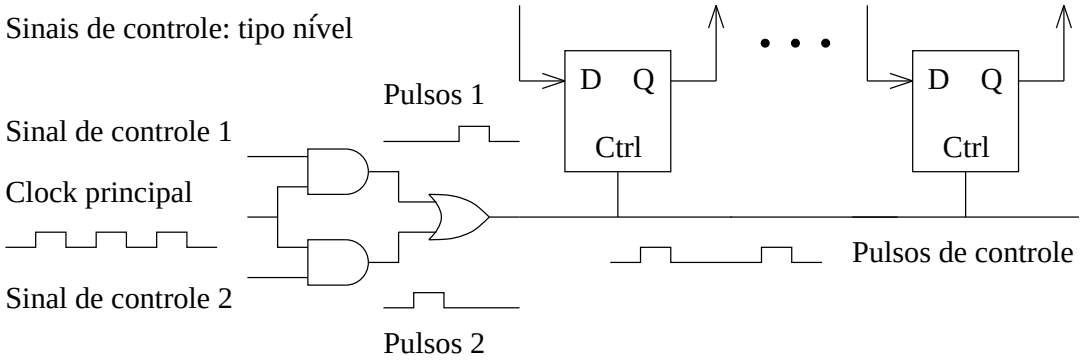


Figura 5.7: Exemplo 2 de controle de sinal de *clock*.

- Em uma grande variedade de aplicações de sistemas digitais, o conteúdo dos elementos de memória ou não é modificado ou é condicionalmente carregado com o resultado da aplicação de alguma função sobre um conjunto de dados.
- Portanto, para tais sistemas, o *flip-flop* do tipo D é o mais utilizado, pois realiza a função de armazenamento com um custo menor do que o *flip-flop* do tipo JK.
- Sinais de controle de CLEAR e PRESET, independentes do sinal de ativação do *flip-flop*, são comumente utilizados.
- Porém, a fim de evitar mudanças impróprias, provocadas pela aplicação de tais sinais ao mesmo tempo em que o *flip-flop* é ativado, tais entradas de controle são normalmente utilizadas apenas para a inicialização (*reset*) do circuito.
- Uma arquitetura do tipo *controlled-clock* comumente encontrada na prática é a denominada Lógica de Transferência entre Registradores (*Register-Transfer Logic* ou RTL).
- Nos circuitos que possuem tal arquitetura, os dados são condicionalmente armazenados em registradores.
- De acordo com o processamento a ser realizado, os dados são transferidos entre registradores específicos.
- Eventualmente, podem ser inseridos circuitos combinacionais no caminho de ligação entre dois registradores, os quais serão responsáveis pela implementação de funções lógicas e/ou aritméticas, necessárias ao processamento dos dados armazenados.
- As transferências são controladas por meio de sinais pulsantes secundários, sincronizados com o sinal de *clock* principal.
- Normalmente, todos os sinais de um sistema são organizados em conjuntos de ligações, denominados de barras ou barramentos: barra de dados (*data bus*), barra de controle (*control bus*) e barra de alimentação (*power bus*).
- A transferência entre dois registradores é realizada por meio de uma barra de dados (*data bus*).
- A Figura 5.8, apresentada em [HP81], ilustra um modelo genérico para circuitos seqüenciais *controlled-clock*.
- O modelo separa o sistema em duas partes: um bloco de processamento de dados e um bloco de controle.
- O bloco de processamento de dados incorpora os registradores que armazenam os dados a serem processados e a lógica combinacional necessária à realização das funções de processamento.
- O bloco de controle representa os circuitos seqüenciais responsáveis por gerar os sinais de controle (níveis e pulsos) que realizam as transferências apropriadas, sincronizadas com o sinal de *clock* principal.
- Normalmente, o número de linhas de entradas de controle e o número de linhas de sinais de controle são pequenos em comparação tanto ao número de linhas de dados de entrada e de saída quanto ao número de linhas de interconexão de dados, internas ao bloco de processamento de dados.

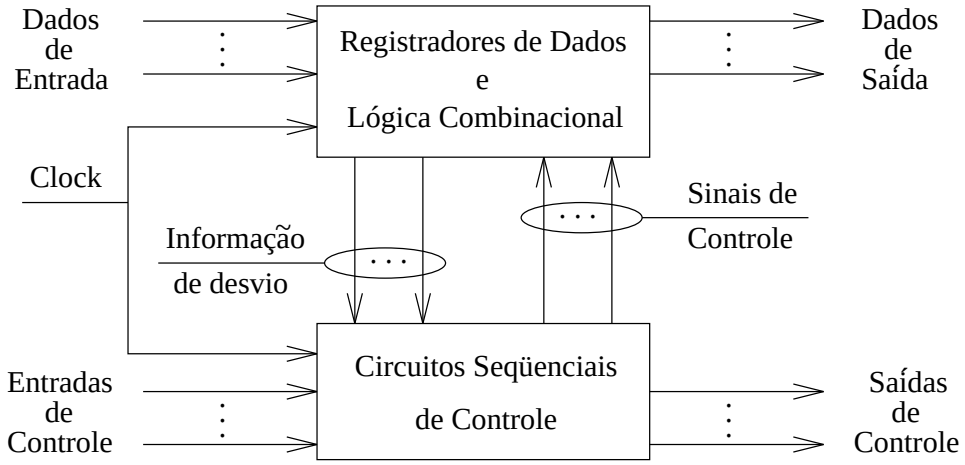


Figura 5.8: Modelo genérico para circuitos seqüenciais *controlled-clock*.

Capítulo 6

Circuitos seqüenciais *level-mode*

6.1 Introdução

- A Figura 6.1 apresenta um modelo genérico para circuitos seqüenciais *level-mode*.

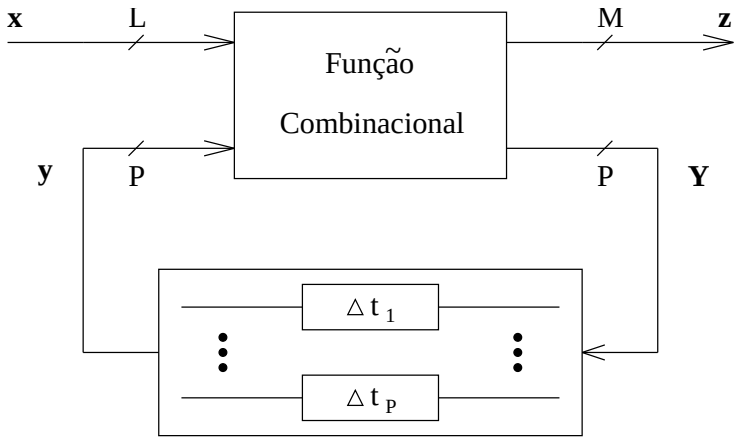


Figura 6.1: Modelo genérico para circuitos seqüenciais *level-mode*.

- O modelo destaca a ausência de elementos de memória permanente.
- Ao invés disso, tal estrutura se utiliza de elementos de memória temporária, implementados através de atrasos.
- Por sua vez, os atrasos que implementam o bloco de memória não são blocos de retardo isolados. Eles representam a concentração de atrasos de propagação existentes no circuito combinacional. Conseqüentemente, os valores de tais atrasos podem variar ao longo do tempo, uma vez que eles serão dependentes dos diversos fluxos que os sinais podem percorrer através do circuito combinacional.
- Assim como nos demais classes: $y_k^{n+1} = f_k(Y_1^n, \cdots, Y_P^n)$, $k = 1, 2, \cdots, R$.
- Mais especificamente, neste caso: $P = R$ e $y_k(t + \Delta t_k) = Y_k(t)$, $k = 1, 2, \cdots, P$.
- Todos os sinais presentes no circuito são do tipo nível.
- Uma mudança de estado é provocada por uma mudança de nível nos sinais de entrada.

- Em resumo, a Figura 6.1 indica que um circuito seqüencial *level-mode* genérico é simplesmente um circuito combinacional realimentado, com entradas do tipo nível.
- Porém, deve ser lembrado que, por definição, todo circuito seqüencial deve ser realimentado, mas a simples realimentação de um circuito combinacional não garante que ele passe a se comportar como um circuito seqüencial.
- Em circuitos seqüenciais pulsados (*pulsed* e *clock-mode*), é natural que as saídas dos elementos de armazenamento sejam escolhidas como variáveis de estado, uma vez que eles são também os elementos de sincronismo do sistema.
- Nos circuitos *level-mode* a realimentação é continuamente aplicada. Assim, qualquer ponto dela pode ser identificado como uma variável de estado sem causar prejuízo à análise ou ao projeto do circuito.
- Diz-se que um circuito opera em modo fundamental se, e somente se, não forem permitidas mudanças nos valores de suas variáveis de entrada até que o circuito atinja um estado estável.
- Deve-se observar que o modo fundamental é uma restrição quanto à forma como o circuito é operado e não quanto ao tipo de projeto executado.
- O modo fundamental pode ser implementado permitindo-se que apenas uma das variáveis de entrada seja modificada por vez e garantindo-se que modificações sucessivas em tais variáveis só ocorram após a estabilização do circuito.

6.2 Problemas comuns em circuitos *level-mode*

- Nos circuitos seqüenciais controlados por pulsos (*pulsed* e *clock-mode*) a realimentação é interrompida pelo bloco de memória e é ativada segundo um certo sincronismo.
- Por outro lado, nos circuitos seqüenciais *level-mode* a realimentação encontra-se ativa durante todo o tempo.
- Conseqüentemente, podem ocorrer instabilidades e incertezas.
- Alguns problemas mais comuns são:
  - As condições de entrada ou de saída de um circuito podem ser indeterminadas.
  - A condição da saída de um circuito pode ser instável, a qual pode apresentar mudanças ainda que as entradas não sejam modificadas.
  - A condição da saída de um circuito, mesmo que estável, pode não ser preditível a partir das condições da entrada.
- As soluções mais empregadas para tais problemas são:
  - Evitar instabilidades crônicas (oscilações): se o circuito exibe oscilações para alguns valores de entrada e é estável para outros, então as condições que imprimem oscilações devem ser evitadas.
  - Evitar incertezas: se o circuito exibe comportamento indeterminado para alguns valores de entrada e determinismo para outros, então as condições que provocam indeterminismo devem ser evitadas.

  - Operar em modo fundamental.
  - Operar em modo pulsado (*pulsed* e *clock-mode*).

- Alguns pontos devem ser ressaltados:

  - Circuitos que exibem oscilação sob certas condições não podem ser utilizados em aplicações de armazenamento ou processamento de dados. Porém, tal comportamento é essencial quando a intenção é gerar sinais de sequenciamento ou temporização.
  - Nem sempre é possível garantir a operação em modo fundamental, uma vez que sinais provenientes de diversas fontes diferentes podem variar aleatoriamente. Nesses casos, uma solução é empregar circuitos sincronizadores extras para garantir a operação em modo fundamental.

6.3 Exemplo de análise de circuito *level-mode*

- Análise de dois circuitos seqüenciais que implementam um *flip-flop* SR.
- Tabela de transição (de estados).
- Tabela de fluxo de estados (*flow table*).
- Tabela de fluxo de estados primitiva (*primitive flow table*).

6.4 Exemplo de projeto de circuito *level-mode*

- Diversas opções de projeto para circuitos seqüenciais que implementem um *flip-flop* SR.
- Definição de uma especificação para um *flip-flop* SR.
- Exemplo de diagrama de tempo.
- Tabela de fluxo de estados primitiva (*primitive flow table*).
- Tabela de fluxo de estados (*flow table*) minimizada ou reduzida.
- Tabela de atribuição de estados.
- Tabela de transição (de estados).
- Síntese das variáveis de excitação e de saída.
- Circuito final.

6.5 Problemas causados pela realimentação contínua

6.5.1 Problemas causados pelo bloco de lógica combinacional

- Existem dois efeitos comuns em circuitos combinacionais: corrida (*race*) e perigo (*hazard*).
- No primeiro caso, após uma mudança nos sinais binários de entrada, espera-se alterar mais de um dos sinais binários de saída. Devido a atrasos internos, os sinais de saída, partindo de um valor inicial (estável), podem assumir configurações intermediárias (instáveis) antes de atingir o seu valor final (estável).
- No segundo caso, após uma mudança nos sinais binários de entrada, duas situações podem ocorrer. Na primeira delas, espera-se que o valor de um determinado sinal binário de saída não seja modificado. Porém, devido a atrasos internos, ainda que o valores inicial e final sejam o mesmo, surgem variações intermediárias. Isso é denominado perigo estático (*static hazard*). Na segunda situação, espera-se que o valor de um determinado sinal binário de saída seja coplementado. Porém, devido a atrasos internos, ainda que o valor final seja o complemento do valor inicial, surgem variações intermediárias. Isso é denominado perigo dinâmico (*dynamic hazard*).
- Para os circuitos combinacionais, embora a ocorrência de valores intermediários não previstos seja inoportuna, uma solução simples é aguardar a estabilização do resultado final.
- Em circuitos seqüenciais pulsados (*pulsed* e *clock-mode*), a ocorrência de configurações intermediárias nas variáveis de excitação também não representa sério problema, uma vez que a realimentação é interrompida pelo bloco que gera e armazena as variáveis de estado. Novamente, uma solução simples é aguardar a estabilização do resultado final.
- Porém, nos circuitos seqüenciais *level-mode*, a realimentação acontece de forma contínua.
- Nesse caso, os valores intermediários não previstos, causados por corridas e/ou perigos no bloco combinacional, geram estados intermediários não previstos, os quais podem provocar mudanças de estado não desejadas, comprometendo o funcionamento do circuito seqüencial.

6.5.2 Problema natural dos circuitos *level-mode*

- Devido à realimentação contínua, os circuitos *level-mode* apresentam um problema envolvendo duas ou mais variáveis de excitação/estado.
- Supondo operação em modo fundamental, após uma variação nos sinais de entrada, uma variável de excitação Y_1 pode sofrer modificação, ser realimentada e atuar sobre uma outra variável de excitação Y_2, antes que a variação da entrada exerça influência sobre Y_2.
- Nesse caso, Y_2 pode assumir um valor não esperado, comprometendo o funcionamento do circuito seqüencial.
- Esse comportamento é denominado de perigo essencial (*essential hazard*).
- Uma vez que o problema é associado ao tipo de estrutura e à sua especificação, ele pode ser detectado diretamente na tabela de fluxo.

- Supondo um sinal de entrada binário, que sofra três variações consecutivas. Caso a primeira e a terceira variações conduzam o circuito aos estados q_1 e q_3, tal que $q_1 \neq q_3$, então existe perigo essencial na tabela de fluxo do circuito [Ung59].

- O comportamento em questão irá ocorrrer se quaisquer duas colunas adjacentes da tabela de fluxo exibirem um dos dois padrões apresentados na Figura 6.2.

(1)	2
3	(2)
(3)	(3)

(a)

(1)	2
4	(2)
	(3)
(4)	3

(b)

Figura 6.2: Padrões de identificação de perigo essencial em tabelas de fluxo.

6.6 Solução para as corridas: atribuição de estados

6.6.1 Definição do problema

Objetivo da atribuição de estados

- Em circuitos seqüenciais pulsados (*pulsed* e *clock-mode*) a escolha da atribuição de estados visa a minimização do bloco de lógica combinacional.

- Em circuitos seqüenciais *level-mode*, operando em modo fundamental, o problema de estados intermediários, causados por corridas no bloco combinacional, pode ser resolvido através de uma atribuição de estados adequada.

- Dependendo da tabela de fluxo em questão, para que se encontre uma atribuição de estados adequada, pode ser necessário aumentar o número de estados do circuito.

Tipos de mudança de estado

- Duas situações podem ocorrer durante uma mudança de estado: i) alteração imediata de estado ou ii) surgimento de estados intermediários (instáveis) não previstos.

- No primeiro caso, as variáveis de estado modificam-se de tal forma que o circuito passa diretamente do estado inicial ao final, sem estados intermediários. Na prática, isso ocorre porque apenas uma das variáveis de estado necessita trocar de valor.

- No segundo caso, duas situações podem ocorrer: ciclo (*cycle*) ou corrida (*race*).

- Um ciclo é definido por uma seqüência única de estados intermediários, instáveis, entre dois estados estáveis (o inicial e o final).

- Os únicos problemas causados pelo ciclo são o prolongamento e a não uniformidade do tempo de estabilização da mudança de estado.

- A corrida caracteriza-se pela existência de diferentes ciclos para um mesmo estado inicial estável. Nesse caso, não é possível prever por qual ciclo o circuito irá fluir.
- Dois tipos de corrida podem ser definidos: não crítica (*non-critical*) e crítica (*critical*).
- Nas corridas não críticas, o estado final estável é sempre o mesmo, independentemente da seqüência de troca das variáveis de estado e, portanto, dos ciclos percorridos. Nesses casos, os problemas são os mesmos dos ciclos.
- Nas corridas críticas, os diferentes ciclos podem levar a diferentes estados finais estáveis. Portanto, corridas críticas representam comportamento não desejado.
- A Figura 6.3 apresenta um quadro resumo das mudanças de estado nos circuitos seqüenciais *level-mode*, operando em modo fundamental.

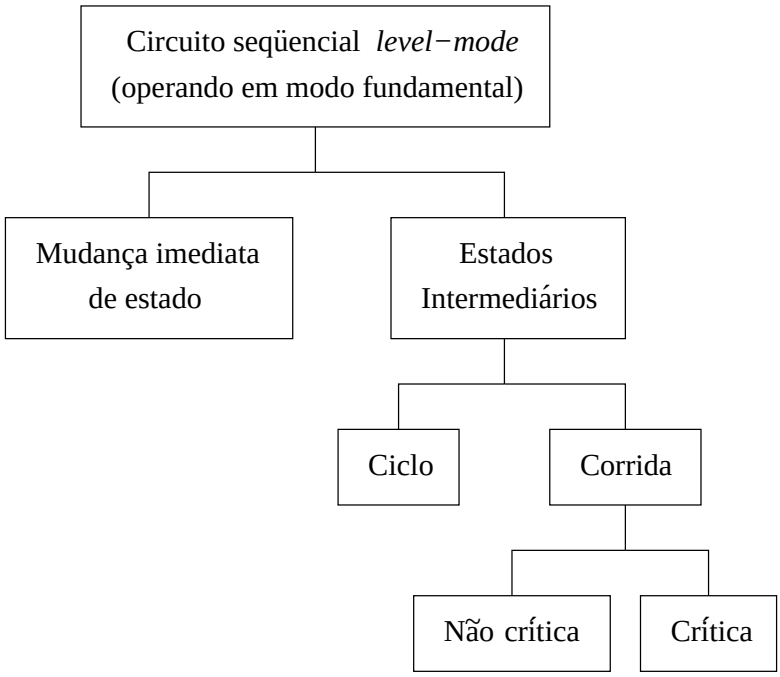


Figura 6.3: Quadro resumo das mudanças de estado nos circuitos seqüenciais *level-mode*, operando em modo fundamental.

6.6.2 Possíveis soluções

Origem do problema

- Em função do que foi exposto, pode-se concluir que: i) uma alteração imediata de estado pode ser interpretada como um caso particular de ciclo e ii) as situações de corrida (ciclos múltiplos) podem acarretar resultados indesejáveis (corrida crítica).
- Portanto, uma solução para o problema de funcionamento indesejado é adotar uma atribuição de estados que realize mudanças de estado apenas por ciclos.
- Para que uma mudança de estado seja executada em ciclo, cada par de estados, do inicial ao final, deve possuir adjacência lógica.
- Dessa forma, para cada mudança de estado, apenas uma variável deverá trocar de valor, evitando a ocorrência de corrida entre as variáveis.

Identificação do problema

- A análise dos tipos de mudanças de estado (ciclos e/ou corridas) que ocorrem em uma dada tabela de fluxo de estados pode ser feita através de um hipercubo booleano.
- Inicialmente, um hipercubo é montado, contendo um número de nós suficiente para conter a quantidade de estados estáveis da tabela de fluxo.
- Em seguida, percorrendo a tabela de fluxo, os estados são associados aos nós do hipercubo.
- A presença de ciclos e/ou corridas é verificada pelas transições presentes no hipercubo.
- Transições realizadas pelas arestas do hipercubo representam ciclos.
- Transições que ocorrem por diagonais significam corridas.
- A classificação das corridas deve ser realizada com o auxílio da tabela de fluxo.
- Supondo-se uma tabela de fluxo organizada de forma que as combinações de entrada definam as colunas, uma transição para uma coluna que contenha apenas um estado estável é associada a uma corrida não crítica. Por outro lado, se a coluna possuir dois ou mais estados estáveis diferentes, a transição representa uma corrida crítica.

Estados reservas (*spare states*)

- A atribuição de estados deve ser feita de tal forma que sejam respeitadas todas as adjacências lógicas em todas as mudanças de estado.
- Dada uma determinada tabela e um determinado número de estados, pode-se não conseguir uma atribuição de estados adequada.
- Neste caso, devem-se empregar estados reservas (*spare states*).
- Para números de estados que não sejam potências de dois, podem-se usar os estados extras como estados reservas.
- Porém, quando o número de estados é uma potência de dois ou não se consegue uma atribuição adequada com os estados extras já existentes, deve-se gerar estados reservas acrescentando-se novas variáveis de estado.

Técnicas de atribuição

- Existem duas técnicas básicas para usar os estados reservas: atribuição por múltiplas linhas (*multiple-row assignment*) e atribuição por linhas compartilhadas (*shared-row assignment*).
- Na técnica de atribuição por múltiplas linhas, aproveitando-se o fato de que o número de estados é dobrado para cada nova variável de estado acrescentada, cada estado original passa a ser representado por duas ou mais linhas na tabela de transição. Esta multiplicidade de representação para cada estado permite que se implemente adjacência lógica para cada par de estados. Conseqüentemente, em qualquer tabela de fluxo, com qualquer número de estados originais, todas as corridas podem ser transformadas em ciclos.
- A técnica de múltiplas linhas necessita que o número de linhas da tabela de transição seja igual a, pelo menos, o dobro do número de estados. Assim, caso o número de estados não seja uma potência de dois, é recomendável que se tente aplicar a técnica de linhas compartilhadas.
- Na técnica de linhas compartilhadas, as combinações reservas de variáveis de estado (linhas da tabela de transição) não são atribuídas a estados individuais. Como o próprio nome já diz, cada linha é compartilhada por diferentes configurações de entrada (colunas da tabela de transição), a fim de transformar corridas em ciclos.

Atribuições tabeladas

- Dois tipos de atribuições de estado podem ser empregadas: universal e padrão [Sau67].
- Atribuições universais são apresentadas em [Sau67], as quais se utilizam de 2 variáveis para 3 estados, 3 variáveis para 4 estados, 4 variáveis para até 8 estados e 5 variáveis para até 12 estados.
- Tais atribuições, ilustradas nas Tabelas 6.1 – 6.4, realizam quaisquer tabelas de fluxo, com os referidos números de estados, sem corridas críticas.
- Dado um determinado número de estados, as atribuições padrões procuram utilizar um número menor de variáveis de estado para representá-los. Porém, elas não são capazes de realizar todas as tabelas com tal número de estados.
- Um exemplo de atribuição padrão para tabelas com 5 estados é apresentado na Tabela 6.5.

		y_1 = 0	y_1 = 1
y_0	0	+	+
	1	+	

Tabela 6.1: Atribuição de estados universal, usando shared-row, para tabelas de 3 estados.

		y_2y_1			
		00	01	11	10
y_0	0	0	2	1	3
	1	1	3	0	2

Tabela 6.2: Atribuição de estados universal, usando multiple-row, para tabelas de 4 estados.

		y_3y_2			
		00	01	11	10
y_1y_0	00	+	+		
	01			+	+
	11	+	+		
	10			+	+

Tabela 6.3: Atribuição de estados universal, usando shared-row, para tabelas de 5 a 8 estados.

		$y_4y_3y_2$							
		000	001	011	010	110	111	101	100
y_1y_0	00	+	+			+	+		
	01			+	+				
	11	+	+						
	10			+	+			+	+

Tabela 6.4: Atribuição de estados universal, usando shared-row, para tabelas de 9 a 12 estados.

		y_2y_1			
		00	01	11	10
y_0	0	+		+	
	1	+	+	+	

Tabela 6.5: Atribuição de estados padrão, usando shared-row, para tabelas de 5 estados.

Conjunto de destinação (*destination set*)

- Conjunto de destinação (*destination set*) é um conceito que se pode utilizar na tentativa de atender a uma determinada tabela de fluxo com uma atribuição que utilize apenas estados reservas já existentes, sem acrescentar uma variável de estado extra.

- Dada uma tabela de fluxo, formam-se conjuntos de destinação para cada configuração das variáveis de entrada (coluna da tabela).

- Para cada coluna, tais conjuntos são formados por um estado estável da coluna com um estado (linha da tabela) que faça transição para o estado estável.

- A fim de que não haja corridas críticas, os membros de cada conjunto de destinação devem ser logicamente adjacentes ou devem ser alocados, em relação aos estados reservas, de forma que as transições cíclicas formadas para todos os conjuntos sejam atendidas sem interferência mútua (cruzamento de ciclos).

Comparações

- Uma comparação entre as duas técnicas pode ser feita com base na complexidade e no tempo de operação do circuito final.

- A técnica de linhas compartilhadas requer um número menor de variáveis de estado. Portanto, o seu uso gera circuitos mais simples.

- A técnica de múltiplas linhas gera transições imediatas. Portanto, o seu emprego produz circuitos com menor tempo de operação.

- Outras técnicas, que reduzem o tempo de operação do circuito, embora demandem maior tempo de projeto e aumento da complexidade do circuito, podem ser encontradas em [Ung69].

6.7 Solução para os perigos

- Dado o perigo estático para o valor binário "1", ele ocorre porque o circuito desativa o mintermo inicial antes de ativar o mintermo final. Dessa forma, acontece a transição $1|_{min\_inicial} \rightarrow 0 \rightarrow 1|_{min\_final}$.

- Portanto, para solucionar o problema, basta acrescentar um mintermo redundante, que permanecerá ativo durante a troca dos mintermos inicial e final. Assim, será realizada a transição $1|_{min\_inicial} \rightarrow 1|_{min\_redundante} \rightarrow 1|_{min\_final}$.

- Adicionalmente, é apresentado em [McC65] o seguinte teorema:
"Um circuito combinacional implementado na forma padrão SOP de segunda ordem que for livre de todos os perigos estáticos para o valor binário "1", será livre de todos os perigos estáticos e dinâmicos.".

- Finalmente, uma forma comum de evitar o perigo essencial é acrescentar atrasos de propagação (inversores em número par) ao circuito.

6.8 Valores das saídas em estados instáveis

- Em mudanças de estados que se fazem por meio de ciclos, deve-se tomar cuidado com os valores atribuídos para as saídas durante os estados instáveis, a fim de se evitar a geração de pulsos espúrios.
- Se, tanto no estado inicial quanto no estado final, o valor especificado para a saída for o mesmo, ele deverá permanecer constante durante o ciclo.
- Se, do estado inicial para o estado final, os valores especificados para a saída forem diferentes, deverá ocorrer apenas uma mudança durante o ciclo. Conseqüentemente, o valor da saída só poderá ser especificado como *don´t care* ('X' ou '—') para um dos estados do ciclo.

Apêndice A

Minimização de tabela de estados

A.1 Introdução

- A minimização do número de estados de um circuito seqüencial pode conduzir à redução da quantidade de circuitos lógicos necessários para implementar os estados (bloco Geração e Armazenamento) e as saídas (bloco Função Combinacional).
- Dada uma tabela de transição de estados (*state table*), pode-se constatar que diferentes estados podem realizar a mesma função.
- Do ponto de vista externo ao circuito, pode-se dizer que não é possível distinguir entre tais estados, uma vez que eles apresentam o mesmo resultado.
- Nesse caso, tal conjunto de estados pode ser representado por um único estado.
- Conseqüentemente, a tabela de transição de estados (*state table*) é simplificada e, possivelmente, o circuito lógico minimizado.
- Na minimização do número de estados de uma máquina seqüencial, a idéia básica é organizar os estados de uma máquina M1 em classes que possuam uma determinada propriedade e, em seguida, definir uma máquina M2, de tal forma que cada estado em M2 cumpra a função de uma das classes em M1.
- Podem-se destacar dois grupos de descrição de máquinas: i) descrição completamente especificada e ii) descrição não completamente especificada.
- Nas máquinas seqüenciais com descrição completamente especificada, utiliza-se o critério de equivalência entre máquinas.
- No caso das máquinas com descrição não completamente especificada, utilizam-se os critérios de compatibilidade e cobertura.
- Pode-se dizer que a equivalência é um caso particular de cobertura, que, por sua vez, é um caso particular de compatibilidade.
- Nas máquinas seqüenciais com descrição completamente especificada, a solução é única e, portanto, o processo é mais simples e direto. Nesses casos, utiliza-se o critério de equivalência para garantir o cumprimento da mesma função por duas máquinas, M1 e M2. Empregando-se as condições de exclusão definidas para equivalência, os estados de uma máquina M1 são organizados em classes disjuntas de equivalência. Para cada classe de M1 é definido um estado equivalente em M2.

- Nas máquinas seqüenciais com descrição não completamente especificada, normalmente deve-se avaliar diferentes soluções possíveis, o que torna o processo mais complexo e menos objetivo. Nesses casos, utiliza-se o critério de cobertura para garantir o cumprimento da mesma função por duas máquinas, M1 e M2. Empregando-se as condições de exclusão definidas para compatibilidade, os estados de uma máquina M1 são organizados em classes conjuntas de compatibilidade máxima. Em seguida, deve-se determinar uma coleção de cobertura (*cover collection*) mínima, que é uma coleção fechada (*closed collection*) mínima que contém cada estado de M1 em, pelo menos, uma classe de compatibilidade. Para cada classe de compatibilidade da coleção de cobertura de M1 é definido um estado de cobertura em M2.

- O formalismo (Definições, Teoremas e Corolários) apresentado nesse capítulo foi retirado integralmente de [HP81].

A.2 Tabelas de estados completamente especificadas

A.2.1 Relações de equivalência

- Quando um par ordenado de elementos (x, y) possui uma propriedade R que os relaciona, pode-se dizer que "x é R-relacionado com y", o que é simbolizado por xRy.

- A relação R é definida como o conjunto de todos os pares ordenados que possuem a propriedade em questão.

- Pode-se assumir que R é uma relação definida sobre um conjunto de elementos, de tal forma que x ou y possam representar qualquer elemento do conjunto.

- Classificação das relações:

  - Reflexão: se xRx é válida para qualquer x, então R é reflexiva.
  - Simetria: se $yRx \leftrightarrow xRy$, então R é simétrica.
  - Transitividade: se (xRy e yRz) $\rightarrow xRz$, então R é transitiva.
  - Equivalência: se R é reflexiva, simétrica e transitiva, então R é uma relação de equivalência.

A.2.2 Estados e circuitos equivalentes

- As tabelas de transição de estados (*state tables*) representam duas funções: a função de próximo estado $\delta(\cdot)$ e a função de saída $\lambda(\cdot)$.

- Pode-se definir a função de próximo estado por: $\delta(q_i^n, x^n) = q_j^{n+1}$.

- Pode-se definir a função de saída por: $\lambda(q_i^n, x^n) = z^n$.

- Para uma seqüência de sinais de entrada dada por $X = x^n x^{n+1} \cdots x^{n+R}$, tem-se: $\delta(q_i^n, x^n x^{n+1} \cdots x^{n+R}) = q_j^{n+(R+1)}$ e $\lambda(q_i^n, x^n x^{n+1} \cdots x^{n+R}) = z^n z^{n+1} \cdots z^{n+R}$.

- Em última análise, dado um estado inicial e uma seqüência de valores de entrada, a função de um circuito seqüencial é produzir uma seqüência de valores de saída apropriada.

- Dessa forma, podem-se estabelecer relações de equivalência entre estados e entre circuitos seqüenciais.

- **Definição 1**: Sejam S e T dois circuitos seqüenciais completamente especificados, sujeitos a seqüências de entrada possíveis e idênticas. Seja $(x^n x^{n+1} \cdots x^{n+R})$ uma seqüência de possíveis valores de entrada, de comprimento arbitrário. Os estados $p \in T$ e $q \in S$ são ditos indistinguíveis (equivalentes), definido por $p \equiv q$, se e somente se $\lambda_T(p^n, x^n x^{n+1} \cdots x^{n+R}) = \lambda_S(q^n, x^n x^{n+1} \cdots x^{n+R})$ para cada possível seqüência de entrada.

- **Definição 2**: Os circuitos seqüenciais S e T são ditos equivalentes, definido por $S \equiv T$, se e somente se para cada estado p em T existe um estado q em S tal que $p \equiv q$, e, inversamente, para cada estado q em S existe um estado p em T tal que $q \equiv p$.

A.2.3 Determinação de classes de estados indistinguíveis

- Uma proposta para se obter a tabela de transição de estados (*state table*) mínima é particioná-la no menor número possível de classes de equivalência de estados indistinguíveis.

- Em seguida, pode-se obter um circuito seqüencial equivalente, onde cada estado corresponda a uma classe do circuito original.

- Uma vez que nem toda partição é uma classe de equivalência, deve-se ter uma forma de se definir corretamente as partições.

- **Teorema 1**: Suponha que os estados de um circuito seqüencial foram particionados em classes disjuntas, onde $p \triangleq q$ denota que os estados p e q pertencem à mesma classe. A partição é composta por classes de equivalência de estados indistinguíveis se e somente se as duas condições seguintes forem satisfeitas por cada par de estados p e q da mesma classe, para cada entrada simples x^n:

  1. $\lambda(p^n, x^n) = \lambda(q^n, x^n)$.
  2. $\delta(p^n, x^n) \triangleq \delta(q^n, x^n)$.

A.2.4 Circuito de classes de equivalência

- Com a tabela de transição de estados (*state table*) particionada em classes de equivalência, pode-se obter um circuito seqüencial equivalente ao original, com o número de estados minimizado.

- **Teorema 2**: Suponha que seja formado um circuito seqüencial T, que corresponda a um circuito completamente especificado S, de forma que para cada estado $p_j \in T$ corresponda uma classe de equivalência $C_j \in S$. O circuito T assim construído, denominado circuito de classes de equivalência, é equivalente a S. Além disso, nenhum outro circuito equivalente a S possuirá um número menor de estados do que T e qualquer circuito equivalente a S que possua o mesmo número de estados de T deve ser T.

A.3 Tabelas de estados não completamente especificadas

A.3.1 Introdução

- Na representação de um circuito digital, a falta de especificação de valores pode surgir por diversos fatores.
- Em circuitos combinacionais, determinadas entradas e/ou saídas dos blocos funcionais podem não ocorrer (*can't happen*) ou podem não importar (*don't care*). Genericamente, ambos os casos são empregados como *don't care*, durante o processo de minimização dos circuitos.
- Em circuitos seqüenciais, as indeterminações podem apresentar várias origens:
  - Nas máquinas completamente especificadas que possuem um número de estados cujo valor não é uma potência de dois, os estados extras da tabela de atribuição podem ser assumidos como *don't cares*. Nesses casos, é comum que eles recebam a denominação de *don't cares* acidentais (*incidental don't cares*).
  - Determinadas seqüências de entrada podem nunca acontecer, gerando indeterminações na tabela de estados (próximas entradas e saídas), as quais podem ser especificadas como *don't cares*.
  - Em máquinas onde as saídas são amostradas em intervalos de tempo maiores do que aqueles das mudancas de estado, podem-se atribuir valores indeterminados às saídas intermediárias, as quais também podem ser assumidas como *don't cares*.
- Em circuitos combinacionais, o valor *don't care* ('X') pode ser substituído apenas por valores booleanos ('0' ou '1'). Nesses casos, a substituição de valores é um processo simples de ser executado e sempre auxilia na minimização das expressões.
- Em circuitos seqüenciais, situações diferentes podem ocorrer:
  - Um valor *don't care* ('X') de estado/saída pode ser substituído por N/M valores de estados/saídas.
  - A substituição indiscriminada de valores de estados/saídas pode: i) proporcionar a minimização do número de estados, ii) conduzir a um número próximo do mínimo ou iii) impedir a minimização.
- Assim, deve-se adotar um método sistemático na tentativa de minimização de estados de máquinas não completamente especificadas.

A.3.2 Noções básicas de compatibilidade

- Dada uma tabela de estados, com próximos estados e/ou saídas não completamente especificados, é possível que se realize combinações de estados, reduzindo o número total estados da tabela.
- Porém, não se pode falar, genericamente, de equivalência de estados em máquinas seqüenciais não completamente especificadas.
- A equivalência entre estados exige que tanto suas saídas quanto seus próximos estados sejam definidos para todos os valores das entradas.

- Tal exigência não é cumprida por máquinas não completamente especificadas.
- Estados não completamente especificados que podem ser combinados em um único estado final são ditos compatíveis entre si.
- Obviamente, estados que são idênticos em seus valores especificados podem ser transformados em estados equivalentes através da atribuição adequada de seus valores não especificados.
- Conseqüentemente, tais estados são compatíveis e podem ser combinados em um único estado.
- Porém, tal condição é suficiente, mas não necessária.
- Estados não idênticos também podem ser combinados, em algumas condições.
- Em qualquer associação de estados, é aplicado o conceito segundo o qual estados que possuem a mesma função dentro do circuito devem produzir os mesmos valores de saída, para os mesmos valores de entrada.
- No caso das máquinas não completamente especificadas, a compatibilidade é associada apenas aos valores especificados de entrada, de próximo estado e de saída.

A.3.3 Formalização dos conceitos de compatibilidade e de cobertura

- **Definição 1**: Seja uma seqüência de valores de entrada $x = \{x^n x^{n+1} \cdots x^{n+(K+1)}\}$, aplicada a um circuito S, cuja descrição é não completamente especificada e que se encontra em um estado inicial q^n. A seqüência x é dita aplicável a q se todos os valores de próximo estado forem especificados, exceto, possivelmente, aquele produzido pela última entrada da seqüência.
- **Definição 2**: Dois estados, p e q, de um circuito S, são ditos compatíveis se e somente se

$$\lambda_S\left(\delta(p, x^n x^{n+1} \cdots x^{n+K}), x^{n+(K+1)}\right) = \lambda_S\left(\delta(q, x^n x^{n+1} \cdots x^{n+K}), x^{n+(K+1)}\right) \ ,$$

  sempre que ambas as saídas forem especificadas, para cada seqüência x aplicável a ambos os estados, onde $x = \{x^n x^{n+1} \cdots x^{n+(K+1)}\}$.
- **Teorema 1**: Se dois estados, p e q, de um circuito S, são compatíveis, então as seguintes condições devem ser satisfeitas para toda entrada simples x:
  1. $\lambda(p^n, x^n) = \lambda(q^n, x^n)$, sempre que ambas forem especificadas.
  2. $\delta(p^n, x^n)$ e $\delta(q^n, x^n)$ são compatíveis, sempre que ambos forem especificados.
- **Definição 3**: Um conjunto de estados S_i, de um circuito S, é denominado uma classe de compatibilidade se cada par de estados em S_i for compatível.
- **Definição 4**: Uma classe de compatibilidade máxima é uma classe de compatibilidade que deixará de sê-la, se um estado que não lhe for pertencente for a ela adicionado. Um estado isolado, que não é compatível com qualquer outro estado, é definido como uma classe de compatibilidade máxima.

- **Definição 5**: Diz-se que um estado p, de uma tabela de estados T, cobre um estado q, de uma tabela de estados S, o que é definido por $p \geq q$, se, para qualquer seqüência de entradas aplicável a q e aplicada a ambas as tabelas, inicialmente nos estados p^n e q^n, respectivamente, as duas seqüências de saídas forem idênticas, sempre que a saída de S for especificada.

- **Definição 6**: Diz-se que uma tabela de estados T cobre uma tabela de estados S se, para estado q em S, existe um estado p em T que cobre q.

- **Teorema 2**: Se um estado p em T cobre ambos os estados q_i e q_j em S, então os estados q_i e q_j devem ser compatíveis.

- **Corolário 2.1**: Se um estado p em T cobre um conjunto de estados S_i de S, então tais estados devem formar uma classe de compatibilidade.

- **Definição 7**: Uma coleção de classes de compatibilidade é dita fechada se, para qualquer classe $\{q_1, q_2, \cdots, q_m\}$ da coleção e para toda entrada simples x, todos os próximos estados especificados, $\delta(q_1^n, x^n)$, $\delta(q_2^n, x^n)$, $\cdots$, $\delta(q_m^n, x^n)$, pertencem a uma única classe da coleção.

- **Teorema 3**: Suponha que, a partir dos n estados de um circuito seqüencial não completamente especificado S, seja formada uma coleção de m classes de compatibilidade, de modo que cada um dos n estados seja membro de, pelo menos, uma das m classes. O circuito S poderá ser coberto por um circuito T, que possua exatamente m estados, $(p_1, p_2, \cdots, p_m)$, de forma que cada classe de compatibilidade de S seja coberta por um dos estados de T, se e somente se a coleção de m classes de compatibilidade de S for fechada.

A.3.4 Sistematização do processo de minimização

- A **Definição 1**, a **Definição 2**, e o **Teorema 1** apresentam as condições de exclusão que podem ser usadas na organização dos estados em classes de compatibilidade.

- A **Definição 3** e a **Definição 4**, fornecem as diretrizes para a geração das classes de compatibilidade.

- A **Definição 5**, a **Definição 6**, o **Teorema 2** e o **Corolário 2.1** tratam da propriedade de cobertura e de sua relação com a propriedade de compatibilidade.

- A **Definição 7**, e o **Teorema 3** indicam as condições de cobertura entre máquinas.

Apêndice B

Introdução à linguagem VHDL

B.1 Introdução

- Desde a implementção do primeiro dispositivo eletrônico em circuito integrado, os avanços tecnológicos têm possibilitado um rápido aumento na quantidade de elementos que podem ser combinados em um único circuito nesse tipo de implementção.
- Naturalmente, com a oferta de uma maior densidade de componentes, a complexidade dos circuitos projetados cresce na mesma taxa.
- Porém, a capacidade de um ser humano em lidar com a idealização, o projeto e a manutenção de sistemas com um grande número de componentes é extremamente limitada.
- Dessa forma, torna-se necessário o uso de ferramentes adequadas a tal tipo de problema.
- Existem duas técnicas de projeto largamente utilizadas na abordagem de problemas de elevada complexidade:
  - Aumentar o nível de abstração na descrição do sistema, de forma que o foco esteja mais na função desempenhada e menos na implementação propriamente dita.
  - Adotar uma visão hierárquica na elaboração do sistema, de forma que, em cada nível de representação, toda a complexidade dos níveis inferiores seja ocultada.
- Nesse sentido, na área de projeto de circuitos integrados, diversas Linguagens de Descrição de *Hardware* (HDL) têm sido propostas, a fim de permitir uma descrição mais abstrata dos componentes e possibilitando que estes sejam organizados de forma hierárquica.
- Uma das linguagens mais utilizadas no projeto de circuitos integrados digitais é a linguagem VHDL (*Very-high-speed integrated-circuit Hardware Description Language*), a qual é apresentada a seguir, de forma introdutória.

B.2 Níveis de abstração

- Físico-matemático
- Componentes
- Células, blocos
- Lógico
- Comportamental

B.3 VHDL como linguagem

- Como qualquer linguagem escrita, VHDL utiliza um conjunto específico de regras que definem aspectos de sintaxe e de semântica.
- Embora seja uma linguagem específica para a descrição de circuitos eletrônicos digitais, VHDL ainda pode ser interpretada como uma linguagem de programação.
- Como tal, ela apresenta elementos comuns a diversas linguagens de programação modernas, alguns dos quais são discutidos a seguir.

B.3.1 Palavras reservadas

- Palavras reservadas (*reserved words* ou *keywords*) são identificadores que possuem um significado especial dentro da linguagem.
- Assim, seu uso é restrito à sua definição original e, uma vez que não podem ser redefinidas, elas não podem ser empregadas para nenhum outro propósito.
- A Figura B.1 apresenta as palavras reservadas de VHDL.

B.3.2 Elementos sintáticos

- Além das palavras reservadas e de identificadores definidos pelo usuário, podem-se utilizar símbolos especiais para escrever o código VHDL.
- Assim como as palavras reservadas, seu uso é restrito à sua definição original e, uma vez que não podem ser redefinidos, eles não podem ser empregados para nenhum outro propósito.
- A Figura B.2 apresenta os símbolos especiais de VHDL.

abs	else	label	package	then
açess	elseif	library	port	to
after	end	linkage	postponed	transport
alias	entity	literal	procedure	type
all	exit	loop	process	
and			pure	unaffected
architecture				units
array	file	map	range	until
assert	for	mod	record	use
attribute	function		register	
			reject	variable
begin	generate	nand	rem	
block	generic	new	report	wait
body	group	next	return	when
buffer	guarded	nor	rol	while
bus		not	ror	with
		null		
case	if	of	select	xor
component	impure	on	severity	xnor
configuration	in	open	shared	
constant	inertial	or	signal	
	inout	others	sla	
disconnect	is	out	sll	
downto			sra	
			srl	
			subtype	

Figura B.1: Palavras reservadas de VHDL.

Símbolo	Significado	Símbolo	Significado
+	Adição ou número positivo	:	Separação variável–tipo
−	Subtração ou número negativo	”	Aspas duplas
/	Divisão	’	Aspas simples ou marca de *tick*
=	Igualdade	**	Exponenciação
<	Menor do que	=>	Seta indicando “então”
>	Maior do que	=>	Seta indicando “recebe”
&	Concatenador	:=	Atribuição de um valor a uma variável
|	Barra vertical	/=	Diferente de
;	Terminador	>=	Maior do que ou igual a
#	Literal incluído	<=	Menor do que ou igual a
(	Parêntese da esquerda	<=	Atribuição de um valor a um sinal
)	Parêntese da direita	<>	Caixa
.	Notação de ponto	−−	Comentário

Figura B.2: Símbolos especiais de VHDL.

B.3.3 Bibliotecas e pacotes

- Em aplicativos de desenvolvimento, é comum que diversos elementos sejam previamente definidos, tais como: identificadores, valores constantes, nomes de variáveis e de estruturas, inicialização de variáveis e de estruturas, macros, funções, objetos.
- O objetivo em se definir previamente tais elementos é facilitar o trabalho do projetista.
- Cada aplicativo possui seus próprios padrões para a organização dos elementos previamente definidos.
- Uma organização simples e bastante utilizada são os arquivos de configuração.
- Por outro lado, uma forma mais estruturada de organização é obtida através do agrupamento de informações em um pacote (*package*) e de pacotes em uma biblioteca (*library*).
- Nos circuitos digitais descritos em VHDL, são largamente utilizados a biblioteca padrão *ieee* e os seus pacotes *standard* e *IEEE 1164*, ambos definidos pelo IEEE (*Institute of Electrical and Electronics Engineers*).

Referências Bibliográficas

[Arm62] D. B. Armstrong. A Programmed Algorithm for Assigning Internal Codes to Sequential Machines. *IRE Transactions on Electronic Computers*, EC 11(4):466–472, August 1962.

[HP81] F. J. Hill and G. R. Peterson. *Introduction to Switching Theory and Logical Design*. John Wiley, New York, NY, 3rd edition, 1981.

[Hum58] W. S. Humphrey. *Switching Circuits with Computer Applications*. McGraw-Hill, New York, NY, 1958.

[McC65] E. J. McCluskey. *Introduction to the Theory of Switching Circuits*. McGraw-Hill, New York, NY, 1965.

[Rhy73] V. T. Rhyne. *Fundamentals of Digital Systems Design*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Sau67] G. A. Saucier. Encoding of Asynchronous Sequential Networks. *IEEE Transactions on Computers*, EC 16(3), 1967.

[Tau82] H. Taub. *Digital Circuits and Microprocessors*. McGraw-Hill, New York, NY, 1982. Em português: McGraw-Hill, Rio de Janeiro, 1984.

[Ung59] S. H. Unger. Hazards and Delays in Asynchronous Sequential Switching Circuits. *IRE Transactions on Circuit Theory*, CT-6(12), 1959.

[Ung69] S. H. Unger. *Asynchronous Sequential Switching Circuits*. John Wiley, New York, NY, 1969.

[Uye02] J. P. Uyemura. *Sistemas Digitais: Uma abordagem integrada*. Thomson Pioneira, São Paulo, SP, 2002.